An FPGA Implementation of Digital Guitar Effects

A Senior Project Report

presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science in Computer Engineering

By

Carson Robles

June 2019

Abstract

One of the most versatile aspects of the electric guitar is its ability to change its sound completely and on-the-fly through the use of effects pedals. Conventional guitar pedals contain one effect and can be chained together. The goal of this project is to serve as a contained multi-effects station with five popular electric guitar effects packed into one product. On top of this, the effects each have two tunable parameters to allow users to dial in the exact tone they are looking for. All of the signal processing done in this project is conducted on an FPGA which also allows the pedal to be reprogrammed at any time to switch out the effects if new effects are desired.

All of the PCB design files and SystemVerilog source code files are publicly available on Github at the following link: https://github.com/carsonrobles/fpga-guitar-pedal.

Table of Contents

Introduction

Motivation

Since the invention of the electric guitar, it has become one of the most popular instruments used in many genres of music; one of the main reasons for this is the large range of sounds that can be produced by using a single instrument. It achieves this versatility through the use of effects pedals, which are essentially devices that are inserted into the signal chain from the guitar to the amplifier that alter the signal passing through them. The project described in this report functions as one of these pedals, but contains five separate effects within it. The motivation for this project is to replace five of the most popular guitar pedals with a single device which is more compact and cheaper than the devices it replaces, while still maintaining the quality that any musician using the device would want.

Project Goals

The main goal of this project is to produce a device that replaces multiple conventional guitar pedals, while maintaining audio quality and providing a familiar user interface to that of conventional gear.

Project Outcomes and Deliverables

The final deliverable of this project is a contained system that can be plugged into a 9V power source and used when a guitar is plugged into one side and an amplifier on the other side. All technologies should be hidden from the user such that no information of the internal system is required to use the product. With this requirement set on the deliverable, the product should be usable by any guitar player that has experience with conventional guitar pedals.

Potential Users

The potential users of this project include all guitarists that use effects pedals with their electric guitars and want to use a more compact system with multiple included effects.

User Needs

The main user needs are great audio quality, desirable effects, and an intuitive user interface. These needs were kept in mind throughout the design and implementation cycles in order to ensure an acceptable and usable end product.

Formal Project Definition

This project strives to implement an FPGA-based multi-effect station for the electric guitar. The final deliverable for this project is a system that takes the guitar signal in as the main input and outputs a processed version of the input as the main output. The system should have five distinct effects each with their own enable. Effects should be enableable in any order and in any combination that the user desires. The FPGA design should fit within the selected FPGA and the power of the system should not exceed 5W as this is the maximum output of the system's voltage regulator.

Spec. Number	Parameter Description	Requirement	Tolerance	Risk	Compliance
1	Number of Effects	5	Min	L	I
2	Power	5W	Max	H	A, T
3	Design Size	Fits in FPGA	Max	H	I

Table 1: Requirements table. I stands for inspection, A for analyzed, and T for tested.

System Design

Overview

The entire project functions as a one-to-one input-output device for the guitar. As such, the project can be thought of as a black box that, in its simplest form, takes the guitar signal as input and outputs the guitar signal with the desired effects applied to it. In order to specify and tune each effect, there are more inputs to the system, such as the effect enable footswitches and the effect tuning knobs, Figure 1 shows this more complete general block diagram.

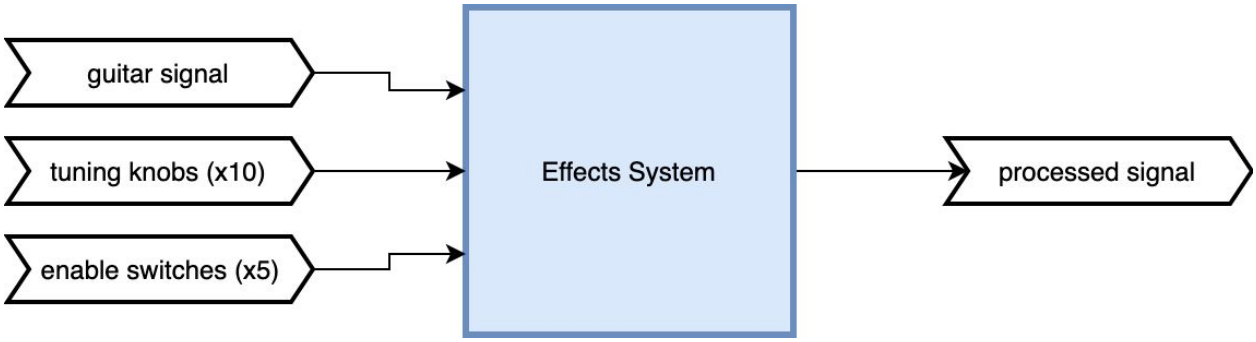

Figure 1: High level system block diagram.

This model is how the user of the product perceives it and is a perfectly fine understanding for someone who wants to integrate the system into their electric guitar setup. As for the details of the internals, the project is broken into two main subsystems: the PCB and the FPGA. The design of the subsystems and each individual effect module will be discussed further in their respective sections of this document.

PCB Design

The custom PCB designed for this project is called the control board; this is because one of its main tasks is to hold the user controls, such as the tuning knobs and enable switches, and to send them to the FPGA. From now on, when addressing this board in the document it will be referred to as the control board. The control board is one of the main two subsystems in this project. Its main functions are to host all the components of the system in an organized manner, provide a steady voltage source to power all the components, and to multiplex the potentiometer voltages for the FPGA to read from its single onboard ADC. All of the components are soldered to the PCB with the exception of the FPGA board and the I2S ADC/DAC module which plug into headers located in the upper right corner on the board so that they can be removed in the future or replaced. This can be seen in the schematic and layout of the control board shown in Appendix 1 at the end of the document.

The first main part of the PCB is the regulated 5V source, the circuitry required to produce this is located in the upper left corner of the board. The board has a female barrel jack

mounted on it that is usually supplied with 9V, as this is the source voltage of many conventional guitar pedals, but it can tolerate up to 26V. This is routed to the 5V linear voltage regulator that produces a steady 5V source. Capacitors are added between the input source and ground and the regulated voltage and ground to smooth the regulated voltage; their values were taken from the regulator's datasheet. All components on the board except the regulator itself are powered off of the 5V regulated source. Figure 2 shows the portion of the schematic that deals with the power source for the project.

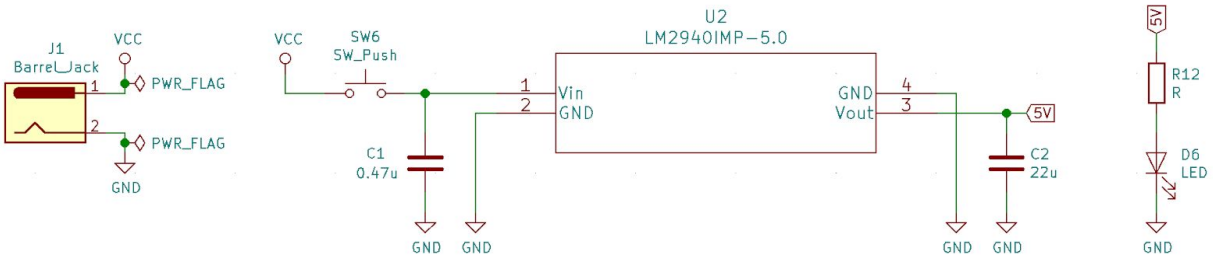

Figure 2: Schematic of the power system on the control board.

The barrel jack, the 5V linear voltage regulator, and the power indicating LED can be seen in Figure 2 from left to right. The voltage regulator has a maximum output current of 1A to accommodate the maximum current that the FPGA board can draw, 600mA, along with an extra 400mA to power the other components as well as add a safety margin.

The components that form the user interface include the ten potentiometers to provide tuning knobs and five footswitches to act as effect enables. The tuning knobs part of the schematic is shown in Figure 3.

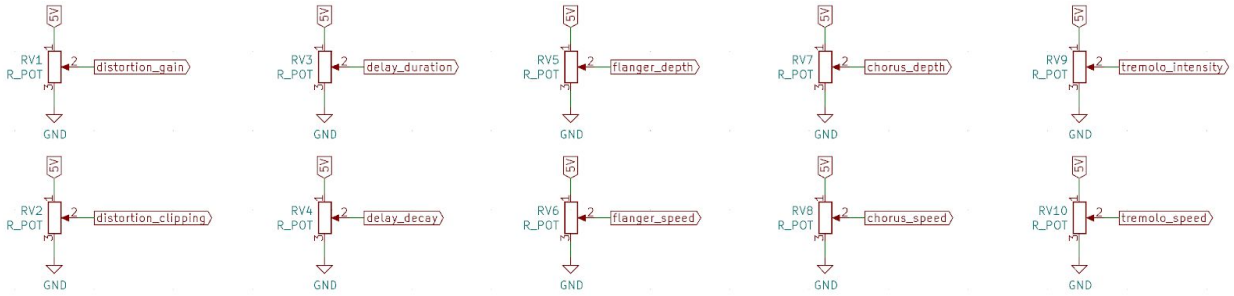

Figure 3: Array of potentiometers on control board used as tuning knobs.

250kΩ potentiometers were used for the tuning knobs, with the first and third pins tied to power and ground, respectively, allowing the middle pin to sweep the entire 0V to 5V range when the shaft is rotated. The analog voltage of the middle represents the current setting on that specific effect tuning. 250kΩ potentiometers were picked simply because a large value was desired and 250kΩ are fairly common in electric guitar gear. While the specific value does not matter significantly to the function because the middle pin will still sweep the same range, it matters in terms of power consumption. Since the first and third pins are attached to power and

ground a constant current of value $I = \frac{V_{supply}}{R_{potentiometer}} = \frac{5V}{R_{potentiometer}}$ flows through the potentiometer at all times. As this current is not useful in any way, minimizing it is ideal when it comes to power dissipation. Looking back at the equation for potentiometer current, we see that the larger the value of the potentiometer the less unnecessary current flows and thus less unused power. For the footswitches, each opens and closes a circuit that drives one of five enable pins on the FPGA. The schematic for this portion of the board is shown in Figure 4.

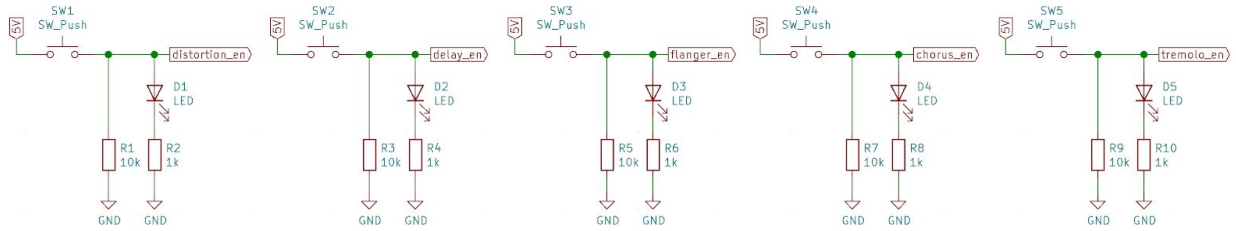

Figure 4: Footswitch enable schematic used for selecting which effects to enable.

Each footswitch circuit includes a pull down resistor, so the voltage on the pin remains at ground when the switch is open, and an LED and resistor in series to signal to the user when the effect is engaged. The FPGA monitors these enable pins and uses them to enable and disable the individual effects modules.

One concern with having ten potentiometers that all need to be read by the FPGA is that in order to read all ten at any given point, ten ADCs are needed, but the FPGA board used only has two onboard ADC channels. The solution to this problem is to use an analog multiplexer. The schematic for the connection between the FPGA and the analog multiplexer is shown in Figure 5.

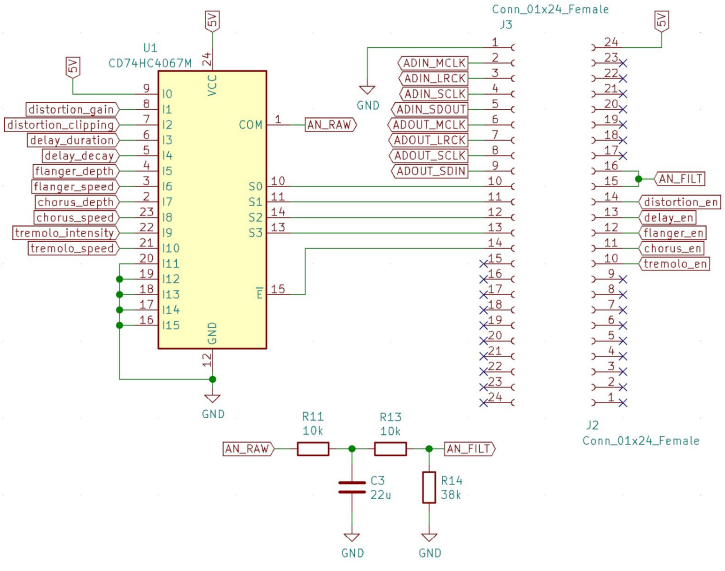

Figure 5: Analog multiplexer and FPGA headers showing how of all 10 potentiometers are routed to the onboard ADC.

All ten potentiometer outputs are routed to their own input on the analog multiplexer (shown on the far left of Figure 5) and only one is allowed to pass to the common output pin. The common output pin is then sent into a resistor-capacitor network which low-pass filters the incoming signal and divides the 0V to 5V range of the potentiometers down to the 0V to 3.3V range that the onboard ADC can handle. The channel is picked by the FPGA which drives the four digital select lines on the analog mux. This signal is then routed to both of the FPGA's ADC channels and the FPGA selects through all of the potentiometer channels and reads each one fast enough for the user to perceive it as all potentiometers are being read continuously. The value is routed to both FPGA ADC channels so that if one of the channels goes bad, the FPGA design can be revised to use the next.

The final part of the control board schematic is the connections between the FPGA and the I2S ADC and DAC module used to sample the guitar signal and output the processed signal. This portion of the schematic is shown in Figure 6.

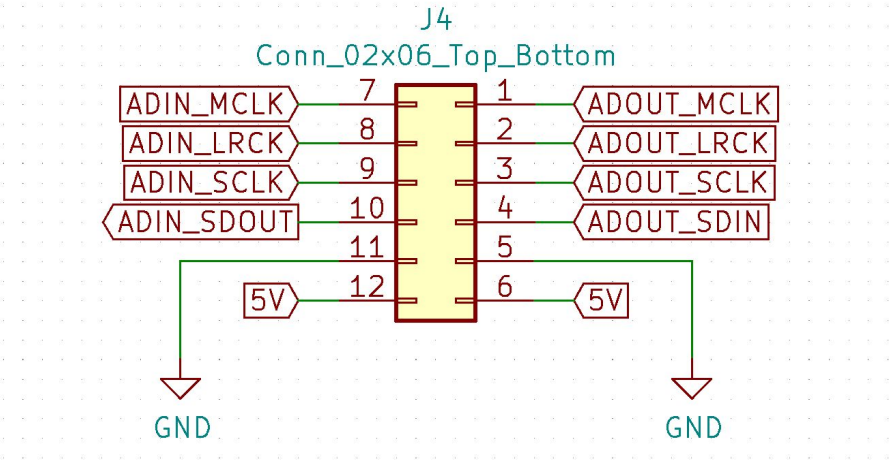

Figure 6: Header for the I2S ADC/DAC module used for sampling the guitar input and driving out the processed signal.

All of the signals connected to the I2S module are directly connected the FPGA header. These signals include the serial data in and out and all of the needed clocks to control the I2S bus operating at a sample rate of 44.1kHz. Generating and using these signals will be covered in more depth in the FPGA Design section.

FPGA Design

The FPGA is the brains of this entire project, all of the signal processing, selecting and tuning of effects is conducted in the FPGA. The FPGA was designed to be a single pipeline consisting of three main blocks: the I2S receiver, the effects pipeline, and the I2S transmitter. A block diagram of this is shown in Figure 7.

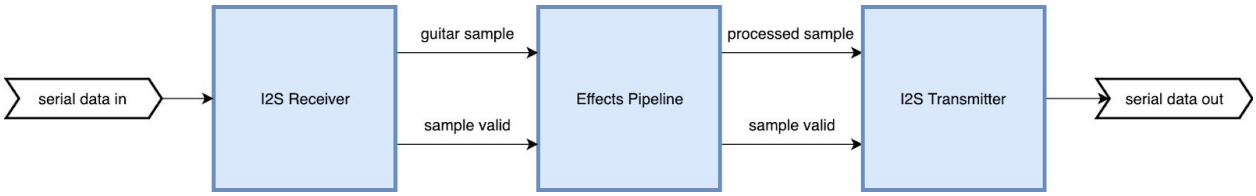

Figure 7: High level block diagram of the FPGA design.

The I2S receiver and transmitter deal with reading and writing to the data converters (the ADC and DAC) in order to receive the guitar signal and output the processed signal. I2S is a serial protocol, commonly used in audio applications. The I2S bus consists of three (or more) lines: the bit clock, the word select clock, and the serial data line. Another common signal to include, and one that is used in this project, is the master clock. The master clock is not a requirement for the I2S bus, but some applications use it to synchronize the ADC and DAC internals.

The serial data in and out are the only data lines in the I2S interface, one bit is placed on the line by the transmitter and read by the receiver, this is synchronized using the bit clock. The bit clock is used to signify the arrival or departure of the next bit of data. For example, the I2S receiver reads the serial data line at the rising edge of the bit clock and the I2S transmitter sends out a new bit on the rising edge of the bit clock. The word clock is used to signify if the incoming or outgoing serial data is intended for the left or right audio channel. This functionality is used for applications with stereo audio, however, guitars use a single audio channel so in this project after both channels are read, only the left channel is passed through the processing blocks. Along with this, the same processed data is written to both the left and right output channels. A timing diagram of an I2S transfer is shown in Figure 8.

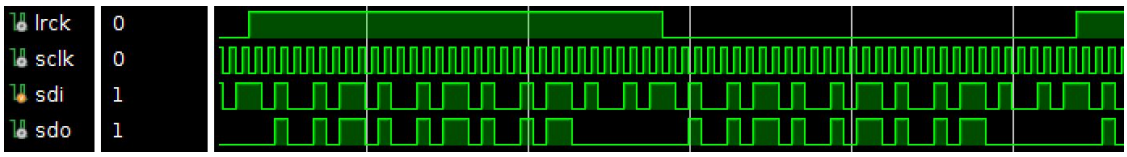

Figure 8: Example timing diagram of and I2S transfer from simulating the module.

In this project, the FPGA generates all of the clocks used in the I2S interface. The frequencies for the master clock, bit clock, and word clock are 22.579MHz, 2.1168MHz, and 44.1kHz respectively. The master clock and bit clock are calculated based on a desired word clock. Since this is an audio processing application, a word clock of 44.1kHz is desirable to be able to effectively sample all frequencies that the human ear can hear. This comes from the fact that the human ear can perceive up to about 22kHz and according to Nyquist-Shannon Sampling Theorem (cited in the References section of this document), the sampling rate must be at least two times that of the highest frequency of the input signal. Since 22kHz is the highest desired frequency to sample, the minimum sampling rate needed is 44kHz; this is where 44.1kHz because it is greater than 44kHz and is a common audio sampling rate used, for example in CD audio. The bit clock is a direct function of the word clock frequency. Since the ADC and DAC used in this project provide 24 bit precision, 48 bits (24 bits for each channel)

must be transmitted and received during one period of the word clock, resulting in 2.1168MHz. The master clock, according the data converter module is 512 times the word clock, which results in a frequency of 22.579MHz.

Once the clocks have been generated, the I2S Rx and Tx cores function as a serial to parallel converter for the Rx and a serializer for the Tx. These consist of two shift registers, one for right and one for left channel. The Rx and Tx modules shift data in and out, respectively, according to the generated clocks.

The next component to the FPGA design is the onboard ADC modules that read from the control board potentiometers and send the values to the effects pipeline. Having effects that are tunable is a huge advantage when it comes to playing the guitar because it allows the user to dial in the exact tone they are looking for. The ADC modules enable this functionality. As explained in the PCB Design section of this document earlier, the FPGA reads from all ten potentiometers by selecting each individually using the analog multiplexer. The FPGA drives the select lines of the analog multiplexer allowing it to pick which potentiometer it wants to read from at any given time. This was implemented as a counter that increments once every millisecond. One millisecond was picked as the time between switches because it is quick enough that the user does not notice the delay from time moving the knob to the tuned parameter actually taking place and it allows enough time for the voltage on the ADC input to settle completely as the low-pass filter does have a transient response that makes the voltage slowly change to the new value when a new potentiometer is selected. Once all ten values have been read, they are sent to the effects pipeline for use in tuning the actual effects modules.

The last big component of the FPGA design is the effects pipeline. This block takes in the effect enables, the tuning parameters, the incoming guitar sample, and a sample valid signal as inputs and outputs the processed sample. This module consists of multiple independent effect blocks which conduct the signal processing and operates to properly connect these effect blocks, enable and disable effect blocks based on the user enable signals, and passes the tuning values to the correct effects. When any individual effect block is enabled it reads the input sample, processes it, outputs it along with a valid flag that it drives high when the processed sample is valid. When disable, effect blocks pass the input sample and valid flag through cleanly to the next effect block. There are five effects included in this project, however, the effects pipeline is designed in a way that makes it easy to add new effects. A block diagram of the effects pipeline is shown in Figure 9.

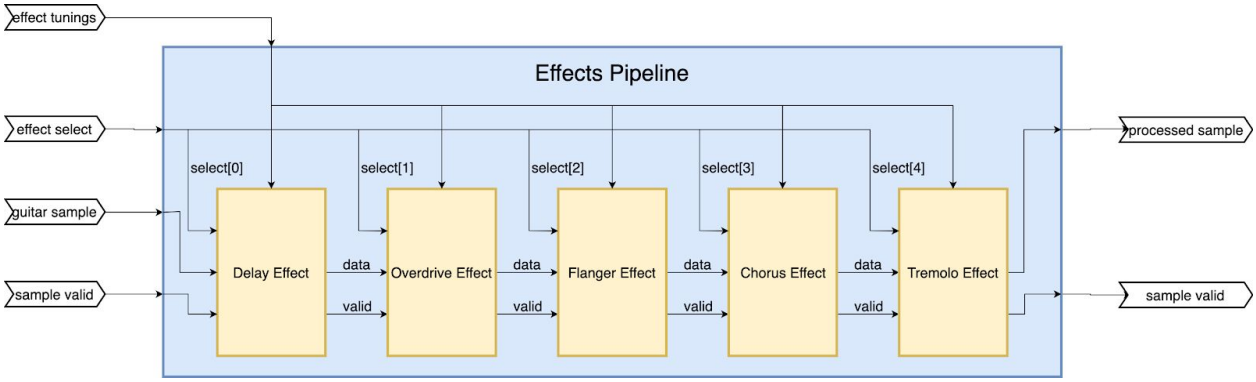

Figure 9: Block diagram of the effects pipeline module.

As shown in the block diagram, this design forms a one way pipe that pushes a new data sample through at a rate of 44.1kHz. Each effect block takes in its tuning parameters, an enable, an incoming sample, and a sample valid flag and outputs a processed sample and a valid flag for this data. One issue that arises from this is the amount of time between samples because as more effects are added it takes more time to pass through the effects pipeline and therefore the time for the valid sample to travel from the I2S receiver to the I2S transmitter increases. Eventually this propagation time will exceed the sampling period and valid samples will no longer reach the end in time. This issue and a possible solution to this problem are discussed in the Future Work section of this report.

Major Design Decisions

During the lifetime of this project, a number of major design choices arose that needed to be decided prior to implementing anything. The most significant included: which FPGA board to use, the layout of the main effects pipeline, which effects to include, and the system to read from 10 potentiometers.

On the FPGA front, some of the deciding factors were: ability to mount to a PCB and be space efficient, have enough BRAMs to buffer samples for effects like delay, have at least one on board ADC channel to read from the potentiometers, and have at least enough IO to drive the I2S ADC and DAC and the analog multiplexer selects and to read from the enable signals. The Digilent CMOD A7 was picked as the board of choice. It is made to fit into a bread board so it was easy to mount headers on the PCB that the FPGA board could slide into. Along with this it has enough BRAMs to buffer samples for a long delay and 48 GPIO pins which is plenty for this project. Lastly, the board has two onboard ADC channels.

The next major decision regarding the FPGA was the layout of the effects pipeline and which effects to include in the project. A goal of this project was to design it in such a way that adding new effects was as simple as instantiating the new module and connecting data and valid signals to the neighboring effects. This became the interface between neighboring effects. All effects adopt this top level structure to make modifications in the future easier. The effects included were inspired from both generally popular effects, such as overdrive and delay, and a few more strange, but interesting effects, such as tremolo, vibrato, and flanger.

The final design choice was how to make the user think the system reads all 10 tuning knobs continuously, using only one ADC. This was accomplished by multiplexing all of the potentiometer outputs onto one common output that was routed to the FPGA ADC channel. Since multiplexers tend to come with channel counts that are powers of two, a 16 channel analog multiplexer was picked for use in this project. The FPGA drives the select lines and counts through the various

Description and Design of Effects

Overview

While the entire system laid out in the previous section is completely necessary for a robust guitar effects station, the effects make up the real function of the product that the user ultimately cares about. This section serves to list the effects included in this project, describe how they work and sound, and how they were designed. As stated before, the effects are what modify the input signal to give the sound a different character or voice. The selected effects included in this project are: overdrive/distortion, delay, flanger, vibrato, and tremolo. General descriptions of these effects along with the design and implementation of these effects will be discussed in more detail in the following subsections.

Overdrive/Distortion

Overdrive is one of the most recognizable and famous effects in electric guitar. Overdrive is anything that adds extra harmonic content to a signal, usually this is accomplished by clipping the incoming signal. Overdrive and distortion are usually said to be the effects that add more “dirt” or “grit” to a clean guitar signal, these sounds are incredibly popular in genres such as Rock and Roll, Metal, and Blues. An example of overdrive applied to a sinusoidal input signal is shown in Figure 11.

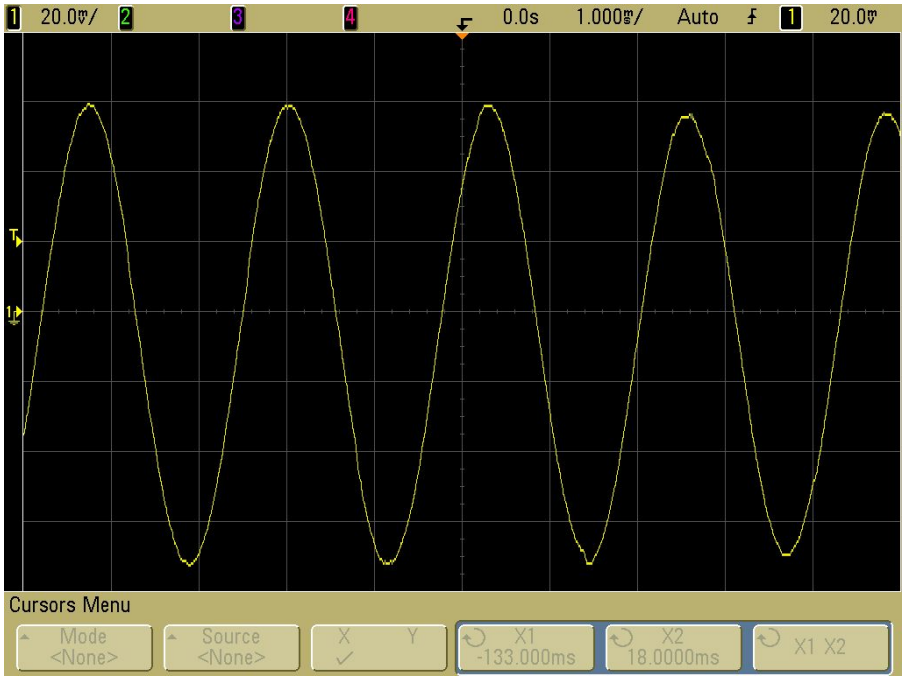

Figure 10: A sinusoidal input signal to provide a reference for Figure 11.

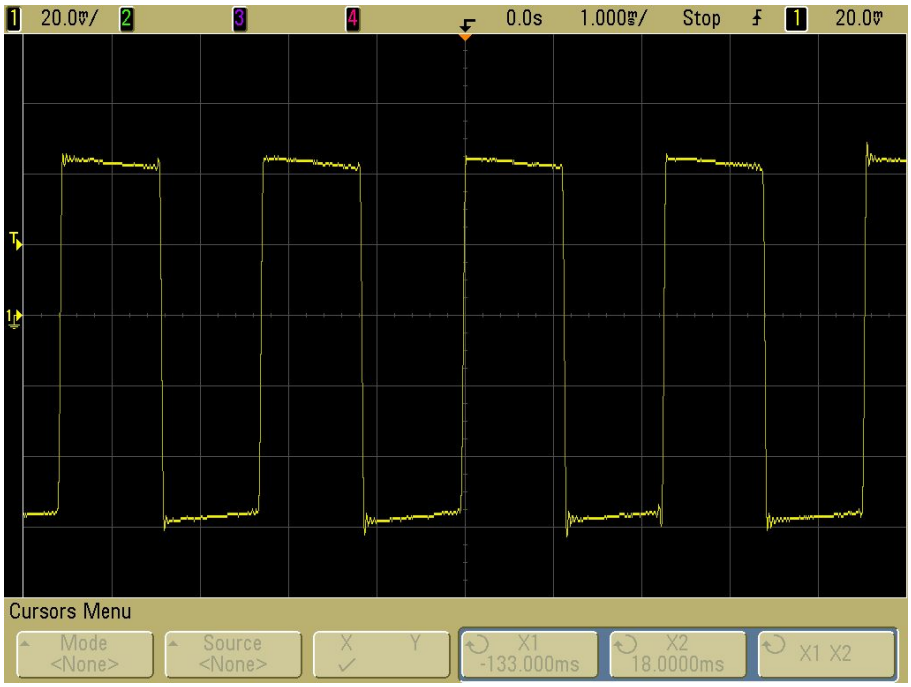

Figure 11: A 440Hz sine wave passed through the overdrive effect.

The overdrive effect implemented in this project uses hard clipping as its way to distort the input signal. The design of this effect is fairly straightforward and it was the simplest of the included effects to implement. A block diagram of this effect is shown in Figure 12.

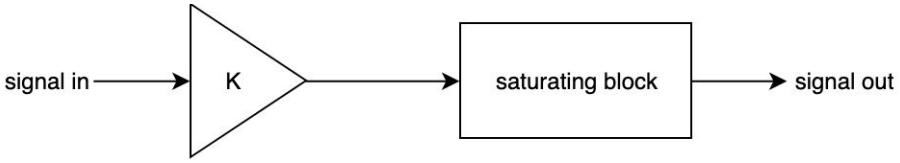

Figure 12: Block diagram of the overdrive effect.

In order to generate this effect, a gain is first applied to the input signal in the form of a multiply set by the gain knob on the pedal; this gain is shown in the block diagram as K. The resulting signal is then compared to the threshold level set by the threshold control knob. If the absolute value of the input signal ever exceeds this threshold the output is held constant at the threshold value, this is done in the saturating block shown in the diagram, named as such because the value can increase or decrease to the threshold and then it saturates at that value. This is implemented using a comparator to compare the input signal to the threshold which is used as the select of a multiplexer to select the input signal with gain to pass to the output or for the threshold value to be passed to the output.

Delay

Delay effects echo the guitar signal back after it has been played along with the current signal. They accomplish this by outputting the sum of the incoming signal with a delayed version of itself. The delayed version is usually decayed or attenuated in order to achieve a stable system. Multiple versions of delay effects exist including: single echo delay, N echo delay, and infinite echo delay. Single and N echo delays can be created using an FIR filter with N delay blocks (the single echo is a special case of the N delay filter where N equals one) that meet at a summing junction. The infinite delay was implemented in this project and is realized using an IIR filter which outputs the input signal summed with the output of a delay line, this output sum is then fed back into the delay line so echoes slowly decay to zero but would echo an infinite number of times if the attenuation was not involved. The block diagram of this effect is shown in Figure 13.

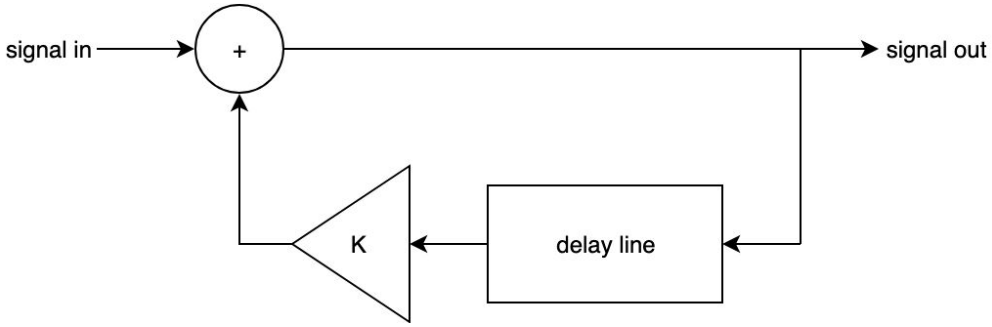

Figure 13: Block diagram of the IIR implementation of a delay effect.

The attenuation block K is adjusted by the user tuning knob and has a range of 0 to 0.5. The depth of the delay line is also adjustable. These two parameters characterize the delay,

changing the value of K decides how loud the echoes are and the depth of the delay line dictates how long it takes for an echo to enter the output.

Flanger

Flanger effects are a bit more difficult to describe how they sound. They make the guitar sound more supernatural and sweeping as the effect is time varying. The effect essentially works by creating a comb filter that moves left and right in the frequency domain sinusoidally and passing the guitar signal through it. Creating such a filter is done by constructing a similar design to a normal delay, consisting of the input signal meeting a delayed version of itself at a summing junction. The difference is that the delay line varies sinusoidally with time, meaning that the amount the signal is delayed is time varying. This is was caused the filter to sweep left and right in the frequency domain and gives the flanger its characteristic sound. The block diagram of the flanger effect implemented in this project is shown in Figure 14.

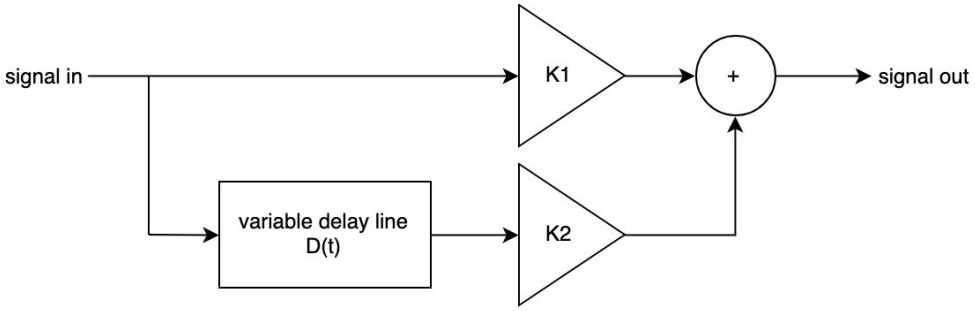

Figure 14: Block diagram of the flanger effect.

It can be seen in the diagram that both the delay line and the input signal both have attenuation blocks; this is to pick how much of the wet and dry signal are summed to create the output. In this context the wet signal refers to the processed components and the dry signal refers to the clean incoming system. By tweaking these gains, the intensity of the effect can be altered. In this project, the two signals have equal weight. A scope capture of the flanger output given a 440Hz input is shown in Figure 15.

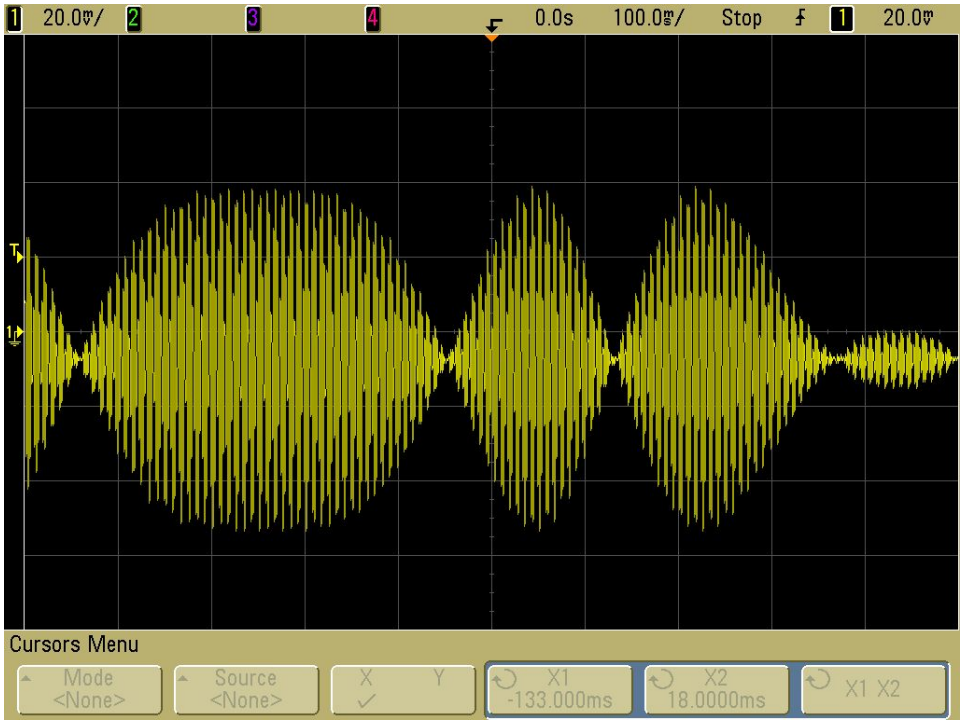

Figure 15: Output of the flanger module with a 440Hz sine wave as input.

Vibrato

The vibrato effect strives to make it sound like the guitarist is bending the string back and forth as they play as it sinusoidally shifts the pitch slightly sharp and slightly flat relative to the actual notes being played. This is accomplished by passing the input signal cleanly through a variable delay block where the depth of the delay line varies sinusoidally with respect to time. The block diagram of this effect can be seen in Figure 16.

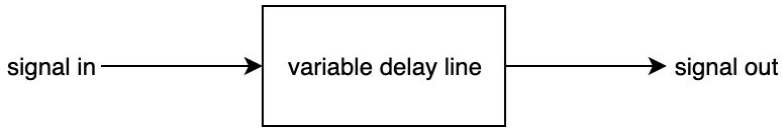

Figure 16: Block diagram of the vibrato effect.

This construct works because passing a signal through a sinusoidally varying delay line creates a simulated Doppler shift on the signal which makes sound as if the frequency increases and decreases with time. This is the exact effect that is necessary to simulate a vibrato.

Tremolo

Tremolo is an amplitude modulation effect: it changes the amplitude (volume) of the signal over time according to some function such as a sine or triangle wave. A triangle wave is

used in this project as it is very easy to generate on board using a counter that counts up to maximum value and then down to zero and back up. Multiplying this function with the input signal then causes the tremolo sound of the volume bouncing up and down. The block diagram of this effect is shown in Figure 17.

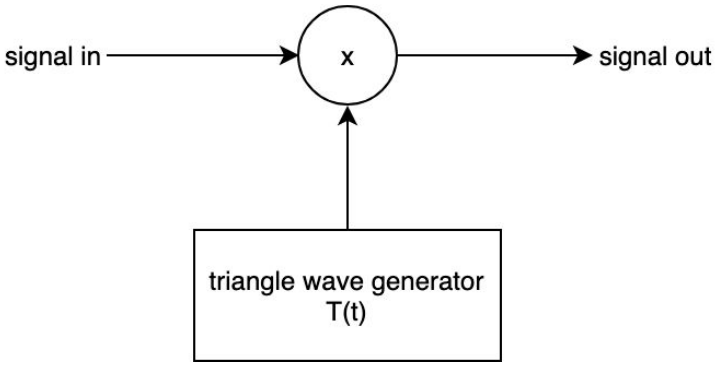

Figure 17: Block diagram of the tremolo effect.

One trick with this effect is that the integer math used in this system gets in the way. Mathematically this problem is solved by creating a triangle wave that ranges from 0 to 1 so that the signal amplitude ranges between 0 and full volume, any higher value causes an unwanted amplification. This is solved by using a counter of 8 bits and multiplying this value by the input signal, the result is then shifted right by 8 bits to effectively divide out this counter leaving the signal within range with the multiply still in effect. A scope capture of the output of this effect can be seen in Figure 18.

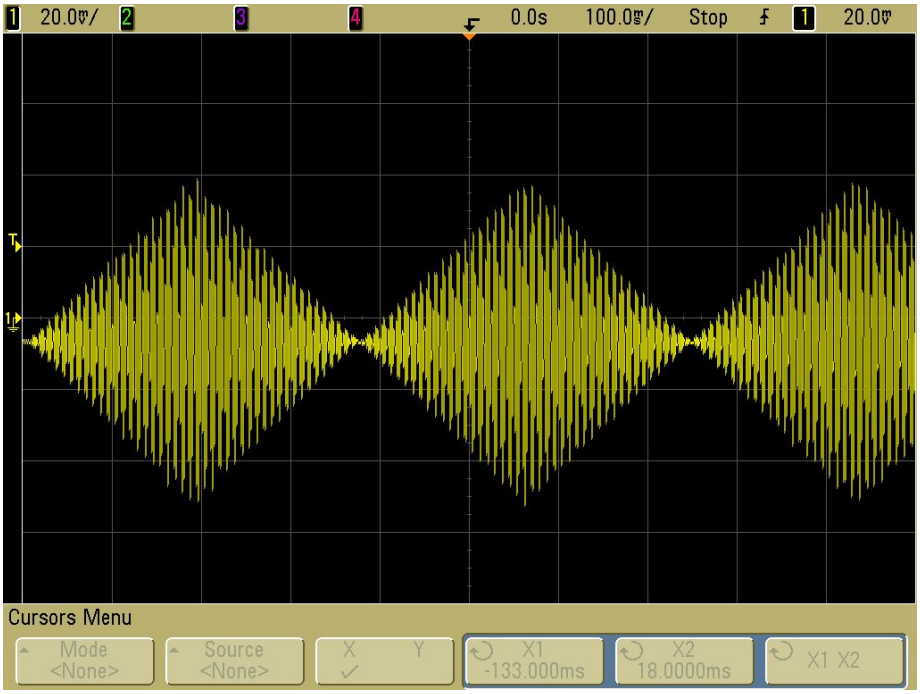

Figure 18: Output of the tremolo effect with a 440Hz sine wave as input.

System Testing and Analysis

Testing and analysis was a big part of this project as it is in any larger FPGA design. All modules in this project should have a corresponding testbench. A general rule that was followed was to first write the testbench and simulate the modules multiple times as it is being designed piece by piece. This helps to catch bugs early and verifies the workings of the module from start to finish. For effects modules another approach built upon the general flow was used. Effects algorithms were first prototyped in Python. Once the prototype had been verified, the algorithm was used to write the RTL for the corresponding FPGA module. This module the went through the simulation design phase. Once verified in simulation, the module was built and placed on the FPGA. The actual system was then analyzed using an oscilloscope to monitor the input and output waveforms and match them to both simulation and the Python prototype. Some scope captures can be seen in the Description and Design of Effects section of this document and an example simulation waveform can be seen in Figure 19.

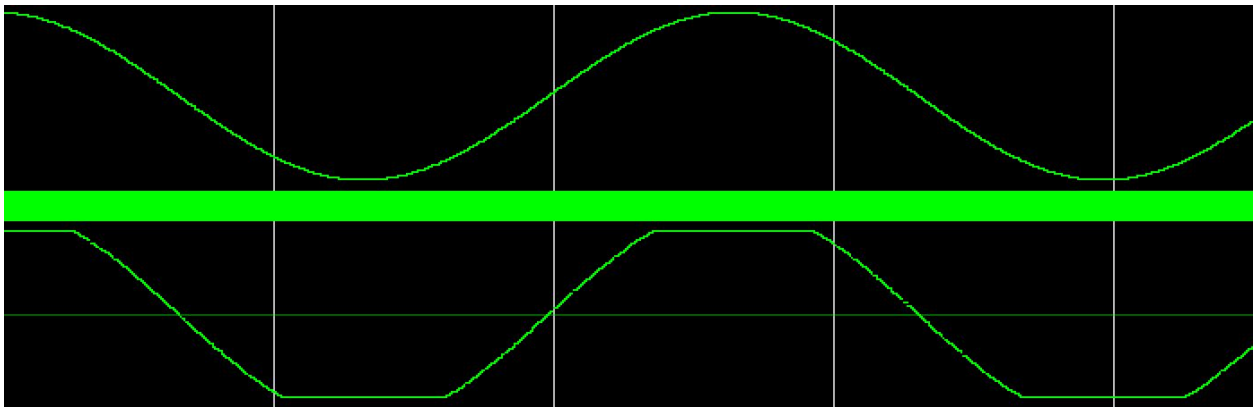

Figure 19: Example simulation waveform from the overdrive effect.

Conclusion and Future Work

As a whole, this project was completed without many major issues that couldn't be solved. The effects work as expected and are fairly noise free, leading to a strong sounding signal of good quality. One area that needs to be fixed in the future is the control board's voltage regulator. A 5V regulator was chosen for this project because the FPGA board uses a VCC of 5V. However, after a long amount of use the GPIO pins used to read the effect enables began to malfunction. This was due to the fact that even though the supply voltage is 5V the IO pins can only tolerate up to 3V3. This was missed during the part selection phase of the design and ended up causing damage to the FPGA board. If this project is continued, this should be one of the first issues to address.

Another improvement that could be made to the design is to make the ADC and DAC clocks more independent, but still synchronized by the master clock. This would fix the problem of running out of time in sampling periods. If the DAC clock rather than running off of the same clocks as the ADC had its clocks slightly phase shifted this would allow for more logic to exist in between the input and output I2S blocks.

These are the main two issues that should be addressed in the future to better the project. In general the project was a success. The original project proposal was to create five effects and this requirement was met and turned out better than expected.

Reflection

Over the last two quarters I have spent a considerable amount of time designing and implementing the system that I just described above. I chose this project because it was the perfect intersection of one of my favorite hobbies--playing guitar--and some of my favorite topics that I've experienced as a Computer Engineering student--FPGA design, PCB design, and digital signal processing. Throughout the course of this project, I have furthered my knowledge in all of these aspects of Electrical and Computer Engineering and learned a ton and the project turned out to be successful.

One of the coolest parts of this project was being able to directly realize many of the subjects that I learned about in my Digital Signal Processing class that I took winter quarter of this year. I had started this project knowing very little about signal processing and how I was actually going to implement the system. At the same time I was starting this class and I don't think I would have gotten nearly as far into the project without the knowledge I gained in this class. Toward the end of the class we had a lecture on digital audio effects, this lecture turned out to be a basis for my second quarter work. We discussed effects such as Tremolo and Vibrato along with others. With this knowledge I understood the signal processing behind these effects and was able to mix it with my digital design knowledge and convert it into a working FPGA design. Actually applying the concepts that I learned in lecture and realizing an actual working system that I could jam on was one of the most satisfying experiences I've had in college.

This project was also a huge learning experience because of how much it taught me about long term projects. I worked on this project pretty consistently throughout the entirety of the last two quarters and I really learned the importance of testing and simulating, making small changes that are thoroughly tested before merging them into the master revision, and keeping a high level of organization and modularity so parts designed a few months ago can be easily picked back up at any point. Without these key points making any large project that spans months will be unmanageable.

This senior project was a great experience in terms of learning and growing as an engineer and resulted in a very fun to make and use project.

Appendix

Appendix 1

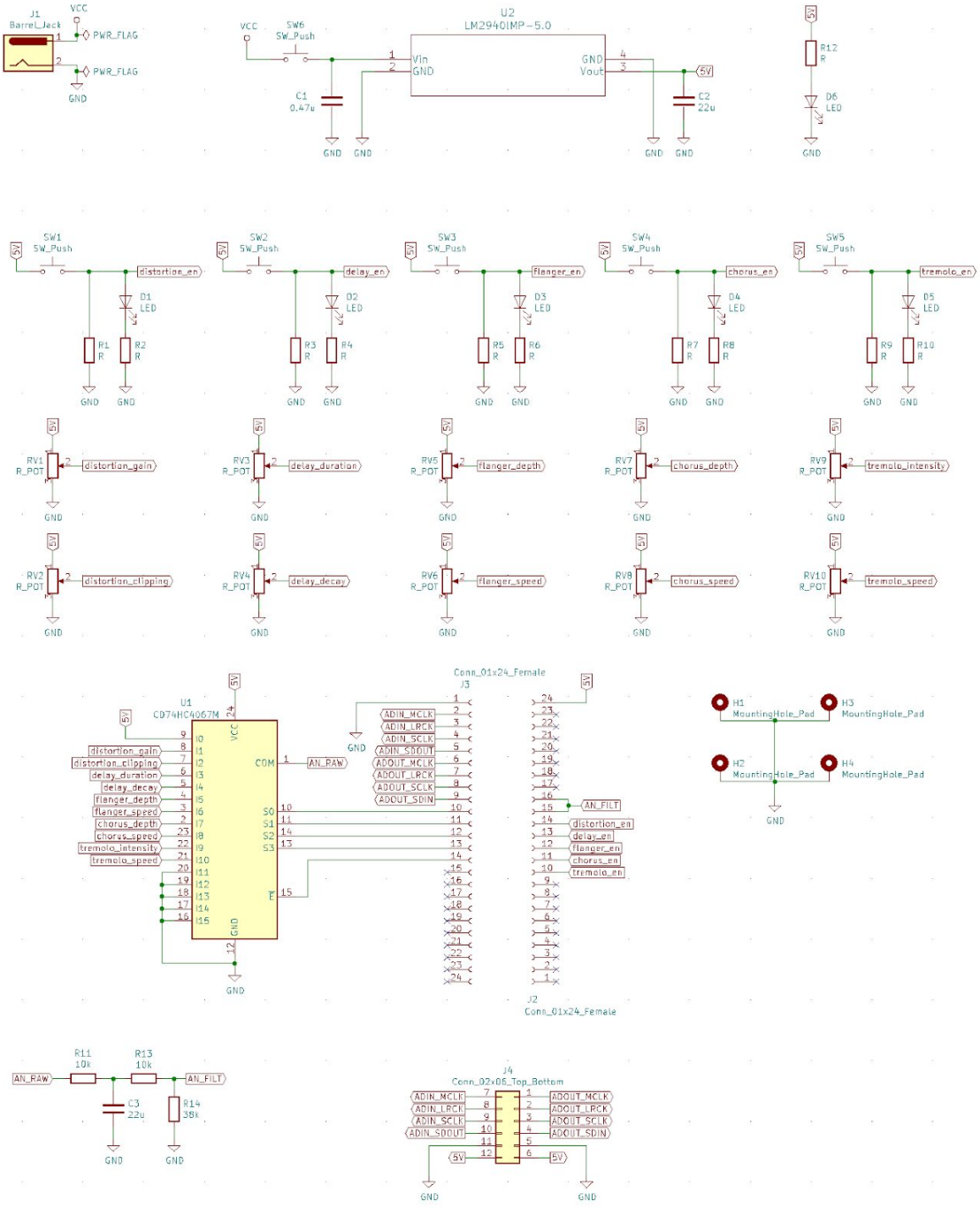

Figure 20: Schematic of the control board.

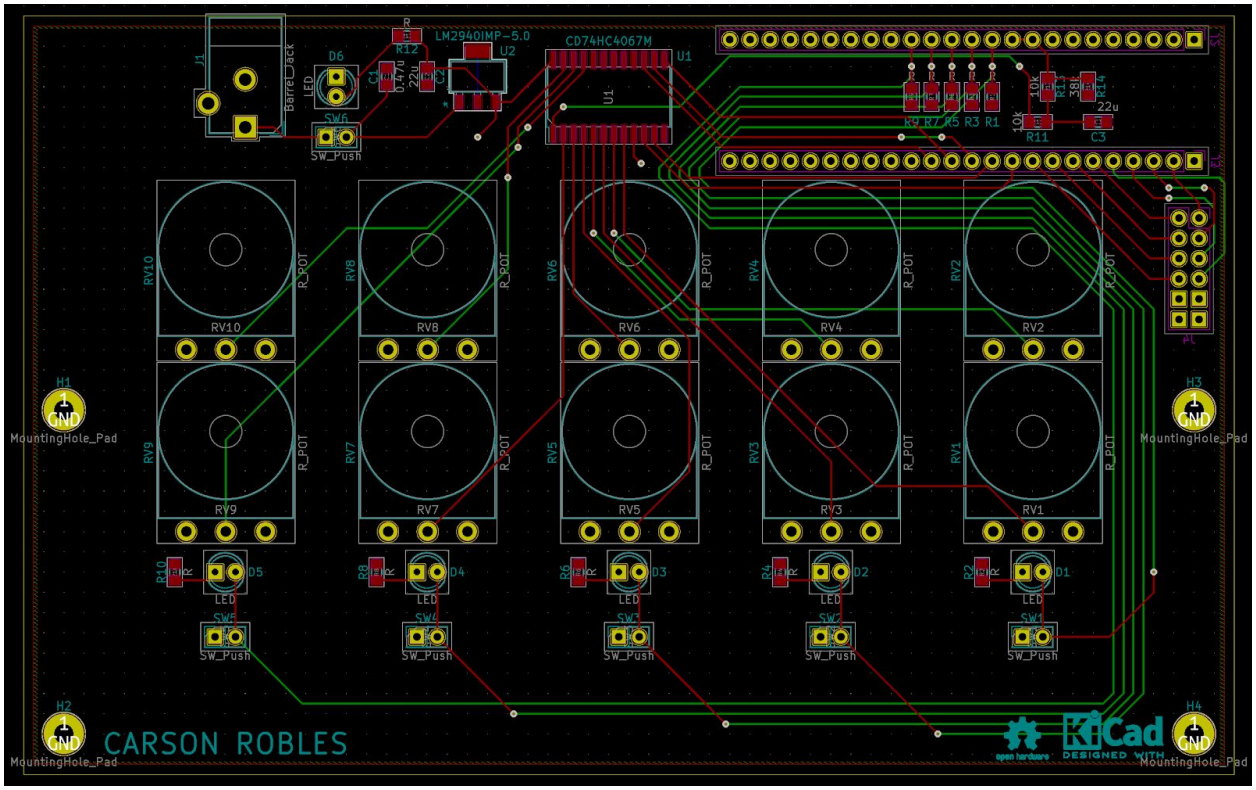

Figure 21: Layout of control board with power and ground planes removed for clarity.

References

[1] ADC/DAC I2S Module Datasheet
https://store.digilentinc.com/pmod-i2s2-stereo-audio-input-and-output/
[2] CMOD A7 Datasheet
https://store.digilentinc.com/cmod-a7-breadboardable-artix-7-fpga-module/
[3] Information on the I2S protocol https://en.wikipedia.org/wiki/I%C2%B2S
[4] Nyquist-Shannon sampling theorem
https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem