DSP Guitar FX Pedal

by

Stephen Rock

Senior Project

ELECTRICAL ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

September 2021

Table of Contents

List of Tables

List of Figures

Acknowledgement

Thank you to my advisor Dr. Wayne Pilkington for his guidance and support throughout the development of this project.

Abstract

A guitar effects (FX) pedal, also referred to as a stomp box, manipulates the electric signal produced by guitars to produce a variety of different sounds for the application of music. Musicians use guitar effects to produce sonically different and interesting sounds for the sake of performance, creation, and art. Although the different types of effects are numerous, the basic effects fall into either distortion, modulation, or delay. The use of guitar pedals allows the musician to accurately perform desired pieces of music or compose new songs.

The current guitar pedal market includes many effects by many manufacturers, however many of the pedals cost over $100 and include a minimal selection of effects. Although there are many ways to manipulate audio signals, Digital Signal Processing offers an inexpensive way to produce multiple effects using minimal amounts of hardware. An inexpensive, good sounding pedal that includes multiple effects would allow more musicians to use effects in their music.

Chapter 1: Introduction

Chapter Overview: Chapter 1 includes an introduction to the DSP Guitar FX Pedal.

Introduction

An electric guitar produces sound through the conversion of the movement of strings into a voltage signal. The use of signal processing circuits for the application of music followed closely to the creation of electronic instruments. A stomp box, a unit containing knobs and switches to adjust the effects, house the signal processing circuitry. With the developments of digital signal processing, a single processing unit could apply multiple effects without the size or expense of analog circuitry.

Motivation

The purpose of this project is to create an effects pedal that will inspire musicians to play guitar and create music. Applying effects to dry guitar signals makes the sound unique and interesting to the listener. Customers will use this product to create and perform music. However, many effects on the market are expensive, ill-sounding, or difficult to use, so the development of a new FX pedal seeks to fix those issues.

Description of Potential Customer

Potential customers include home, gigging, and recording musicians. The home musician plays their instrument in a single location, either for personal enjoyment or learning an instrument. The relatively stationary nature of the home musician means the gear used does not move on regular basis. The gigging musician plays shows at venues that vary greatly from concert halls to bars. The gigging musician performs music in front of an audience, either performing original pieces of music or covering existing songs for artistic and entertainment value. Recording artists record tracks in a studio setting.

Customer Needs

Musicians compose and preform music [2], so the project must directly contribute positively to both music composition and performance. Ideally, the sounds produced by the FX pedal inspire musicians to create music. The other aspect of how it sounds, specifically regarding gigging musicians, the pedal must emulate pre-existing sounds found in music.

The cost of the FX pedal should be kept as low as possible, to give the customer the most amount of effects, and therefore tonal adjustability, per dollar. As guitarists often use a pedal board with multiple stomp boxes, the FX pedal must physically fit on a pedal board and work in the middle of an effects pedal chain. The customer expects certain controls to adjust sound due to the established market of effects pedals, so footswitches should control what effects are in use and knobs should control the different aspects of effects. Because the musician will travel with the FX pedal between venues, the pedal must be durable and hold up to the abuse of gigging.

Context

As a stomp box, this project caters toward live sound solutions. This pedal would operate in live music venues, such as concert halls, restaurants, amphitheaters, bars, outdoor venues, etc. The pedal should be able to operate with a customer's extended pedal board and guitar rig (the equipment a guitarist uses including but not limited to a guitar, pedals, and an amplifier), to apply other effects to the sound not included in this pedal. However, musicians still use stomp boxes in home and recording sessions of music practice and performance, but regardless of the location the stomp box must integrate with the rest of the guitarist rig.

Alternative Solution (Competition) and What sets this Project Apart

Other manufacturers make guitar effects pedals, but few make multi-effect pedals that include different kinds of effects. The leading company in DSP effects pedals is Strymon, but each individual pedal cost greater than $300. Boss also makes reasonably priced effect pedals but does not have multi effect pedals in the $100 range.

Chapter 2: Project Design Engineering Requirements

Chapter Overview: Chapter 2 includes a customer needs assessment, project requirements and specification, and a table outlining the DSP Guitar FX Pedal requirements and specifications.

Functional and Feature Requirements

The application of the guitar FX pedal justifies the requirements for the project. As listed in Table I, the requirements detail normal market standard features such as 9 Volt power and pedal board compatibility, while also including the requirements that fit the goal of the project, making an inexpensive, multi-effects platform.

Performance Specifications

Table I details the specifications for the DSP Guitar FX Pedal and relates each specification to a corresponding requirement.

TABLE I

GUITAR MULTI-EFFECTS PEDAL REQUIREMENTS AND SPECIFICATIONS

Marketing Requirements	Engineering Specifications	Justification
6	1. Device is powered by 9 Volt DC power supply	Most pedals currently on the market use 9 Volt DC power
1	2. Pedal includes multiple selectable effects	Multiple effects increase flexibility and adjustability for the desired sound
2	3. Pedal uses knobs, switches, and footswitches to control selecting the effects.	For live performances, the pedal must be fully controllable without use of an external device such as a laptop
8	4. Small processing audio effects time	The pedal must react quickly to allow instant feedback for the player and to maintain in time for performing songs
4	5. Total pedal does not exceed the dimensions of: # in x # in x # in	This size allows the pedal to fit on a pedal board and allow use of other gear and effects
10	6. Pedal must hold up to abuse of possible dropping and spilled contaminants, along with the normal wear and tear of use (being stepped on)	The pedal must continually work within the possible environments of use and withstand to transportation
7	7. Pedal retails for around $100 to $150	Most single effect pedals on the market have a competitive price point of around $100

Marketing Requirements

1. Multiple built-in effects
2. Simple controls
3. Fits on pedal board (size)
4. Top mounted jacks for easy pedal board wiring
5. Standard 9 Volt power
6. Cost Effective (Number of FX per dollar)
7. Responsive/Reactive to playing
8. Non-noisy standby mode
9. Durable

Functional Decomposition (Level 0)

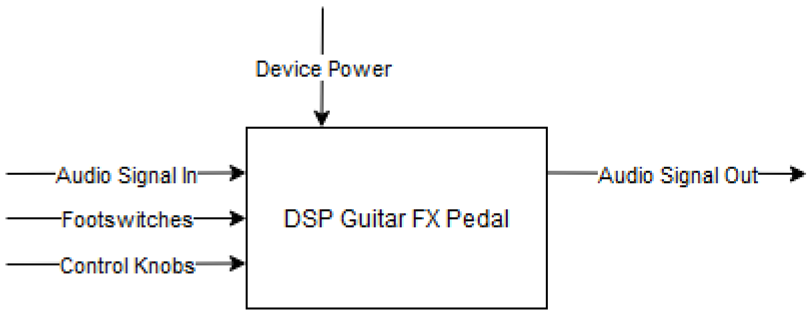


FIGURE I: DSP GUITAR FX PEDAL LEVEL 0 BLOCK DIAGRAM

TABLE II
DSP GUITAR FX PEDAL FUNCTIONS (LEVEL 0)

Module	Guitar FX Pedal
Input	**Audio Signal In** • Input audio signal from guitar or leading effects pedals. **Footswitches** • Switches to toggle activation of applied effects. **Control Knobs** • Knobs modify aspects of the signal processing effects to change the sound. **Device Power** • Powers components withing the pedal
Output	**Audio Signal Out** • Modified audio signal to amplifier or following effects pedals.
Functionality	The DSP Guitar FX Pedal manipulates the Audio Signal in with audio effects controlled with the footswitches and control knobs and outputs the signal to audio signal out.

User Interface

Customers will interact with the product through a combination of knobs and foot switches. The foot switches will control whether audio processing effects will be on or off. The control knobs will vary factors of the audio processing, such as volume, clipping level, or frequency of modulation effects.

Chapter 3: Background

At this project’s most basic form, the purpose of a guitar pedal is to modify the sound of the instrument. The electric guitar produces sound through an electric signal run into an amplifier and a speaker. The vibration of guitar strings over pickups, wound coils of wire around a magnet, causes an electric signal. This electric signal typically produces a clean electric audio signal. However, in music, musicians use effects to modify the signal to produce more interesting sounds. While many effects exist, there are a few overall categories. Distortion or overdrive replicate the sound of pushing amplifiers or other hardware to the maximum rail voltage it can support. Modulation manipulates the waveform by modulating the signal with some other waveform, such as amplitude modulation to produce the tremolo effect. Reverb and delay echo the original guitar signal, either in set intervals of time and decay, or to replicate the acoustics of a room. While many other effects exist, they will not be covered in the scope of this project.

A stomp box describes a device for containing these effects. Often referred to as guitar pedals, stomp boxes contain the circuitry to manipulate incoming signal with the desired effect. The musician controls the effect through a series of knobs, buttons, and switches, and activates or deactivates the effect with a main switch. The pedals sit on the floor so the guitarist can operate the effects while having both hands occupied by the guitar.

The effects are either achieved through analog circuitry or digital processing. Analog circuitry uses analog filters, amplifiers, and other circuitry to produce the audio effect. Digital processing uses digital implementation of microprocessors, digital signal processing chips or FPGA implementation. Digital processing requires conversion of the analog signal to digital values, manipulates the signal through programmed algorithms, then converts the digital signal back to an analog waveform to be played by the amplifier and speaker.

Chapter 4: System Design

Functional Decomposition (Level 1)

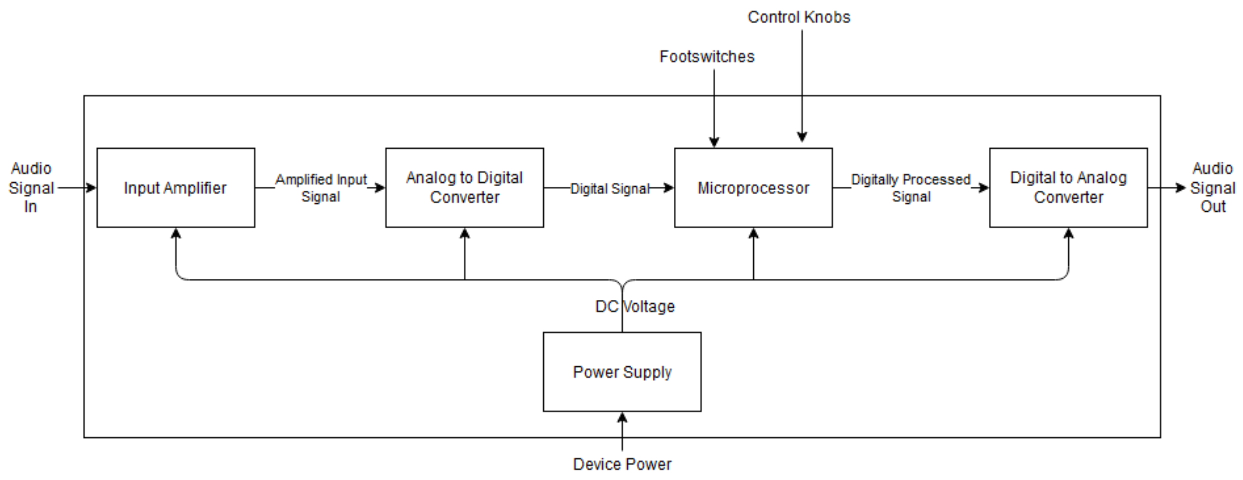


FIGURE II: DSP GUITAR FX PEDAL LEVEL 1 BLOCK DIAGRAM

TABLE III
DSP GUITAR FX PEDAL (LEVEL 1)

Module	**Input Amplifier**
Input	**Audio Signal In** • Guitar input signal: < 0.8 V peak
Output	**Amplifier Input Signal** • Amplified guitar signal: 1.5 V peak
Functionality	The input Amplifier amplifies the input guitar signal to a larger peak voltage so the Analog to Digital Converter reads the signal with better resolution.
Module	**Analog to Digital Converter**
Input	**Amplifier Input Signal** • Amplified guitar signal: 1.5 V peak
Output	**Digital Signal** • Digital representation of input signal
Functionality	The Analog to Digital Converter read the amplified guitar signal and converts the signal to digital representation such that the signal can be digitally processed.

Module	**Microprocessor**
Input	**Digital Signal** • Digital representation of input signal **Footswitches** • Digital inputs to select effect **Control knobs** • Variable inputs to determine effect values
Output	**Digitally Processed Signal** • Output digital signal post processing
Functionality	The microprocessor takes in the digital inputs and manipulates the signal. The effects are applied at this stage.
Module	**Digital to Analog Converter**
Input	**Digitally Processed Signal** • Output signal post processing
Output	**Audio Signal Out** • Analog output signal to Guitar Amplifier or other Pedals
Functionality	The Digital to Analog Converter takes the digital signal produced by the microprocessor and converts the values to an analog signal.
Module	**Power Supply**
Input	**Device Power** • Wall or battery power 9 V
Output	**DC Voltage** • 5 Volt power for circuitry and microprocessor
Functionality	The power supply takes in external power and supplies the circuitry and microprocessor with 5 V.

Chapter 5: Technology Choices and Design Approach Alternatives Considered

Signal Processing Implementation

For this project, I decided to implement the audio signal processing by use of a DSP Chip. Analog circuitry, FPGA implementation, or use of a microprocessor also can produce audio effects. For analog implementation, each audio effect takes its own designed circuit, which both extends design time and final size of the pedal if multiple effects are implemented. With Digital Processing, FPGA implementation has longer development time in comparison to microprocessors or DSP chips. Microprocessor/DSP chip implementation has the quickest development time, relatively low costs, and easy ability for post development product support. My comfort with C programming over Verilog also contributed to the use of a DSP chip. The decision for a DSP chip over a microprocessor considered the application of signal processing, and therefore decided the DSP chip is better suited for processing than a generic microcontroller.

Software

For testing the signal processing algorithms to be implemented in the final build of the pedal, MATLAB, specifically with the addition of the Audio Toolbox, works optimally for this process. MATLAB and the Audio Toolbox allows for real time processing from sampled audio files and allows for a computer without extra hardware to run the algorithms under test.

The compiler selected is Code Composer Studio (CCS) running C as the programming language. The use of the C2000 Microcontroller Launchpad Development Kit by Texas Instruments heavily influenced the use of CCS for best compatibility between controller and compiler.

Components

The main components for digital signal processing are an analog-to-digital converter (ADC), the DSP chip, and a digital-to-analog converter (DAC).

Due to time restraints and minimal manpower, I decided to use Texas Instrument’s C2000 Microcontroller Launchpad Development Kit LAUNCHXL-F28379D that utilizes the TMS320F28379D microprocessor. The effort required to put together custom hardware with a hand selected chip and ADC and DAC would not result in a completed project. The C2000 line of processors is specifically designed for real time processing, Including 200MHz CPUs, 1 MB Flash, and 12-bit ADCs and 12-bit DACs. All components easily operate at above 44.1 kHz required for standard audio sampling.

While other development boards from Analog Devices were considered, the TI development was selected due to its price, clock speed, integrated components, and compiler compatibility.

Chapter 6: Project Design Description

Functional Decomposition (Level 2)

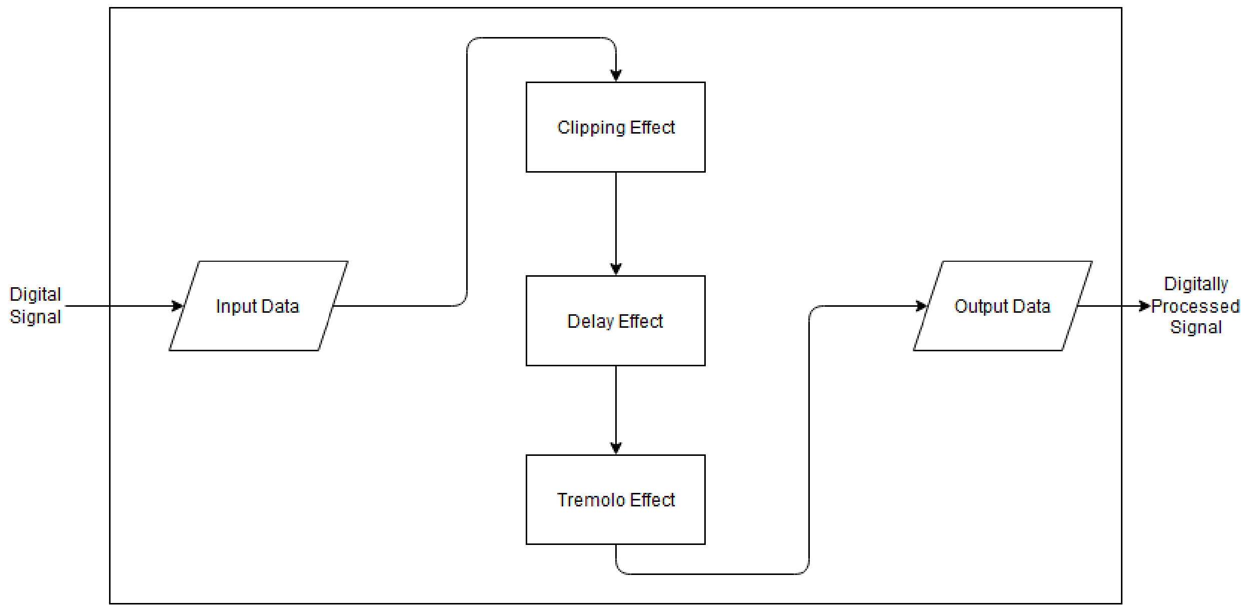


FIGURE III: DSP GUITAR FX PEDAL SOFTWARE LEVEL 2 BLOCK DIAGRAM

TABLE IV
DSP GUITAR FX PEDAL (LEVEL 2)

Module	**Clipping Effect**
Input	**Input Data** • Digital Signal input from ADC
Output	**Digital Signal** • Post Algorithm data
Functionality	The clipping effect takes in the input digital signal and limits the maximum and minimum value of the signal. This is a distortion effect.
Module	**Delay Effect**
Input	**Input Data** • Digital Signal input previous algorithm
Output	**Digital Signal** • Post Algorithm data
Functionality	The delay effect takes in the digital signal values and stores the values in a buffer to output past and current values.

Module	**Tremolo Effect**
Input	**Input Data** • Digital Signal input previous algorithm
Output	**Digital Signal** • Post Algorithm data
Functionality	The tremolo effect modulates the amplitude of the input signal to adjust the volume of the signal. This is a modulation effect.

The input amplifier is an inverting op amp configuration with offset to put the input range from 0 to 3 volts for the ADC. Figure XII shows the schematic of the amplifier. The gain of the amplifier is designed to be around 1.5.

Chapter 7: Physical Construction and Integration

Physical Layout

The physical layout of the of the prototype platform includes the Launchpad development board connected to the input amplifier circuit and audio jacks laid out on a breadboard. The Launchpad Board includes the Analog to Digital Converter and the Digital to Analog Converter along with the 5 Volt power supply and pinouts needed for inputs and outputs.

Packaging

The prototype packaging did not end up matching the initially proposed stomp box packaging enclosed in a metal box with appropriate knobs and foot switches exposed due to time constraints. The current packaging involves the exposed development board and bread board with the amplifier circuits and audio jacks as shown in Figure IV.

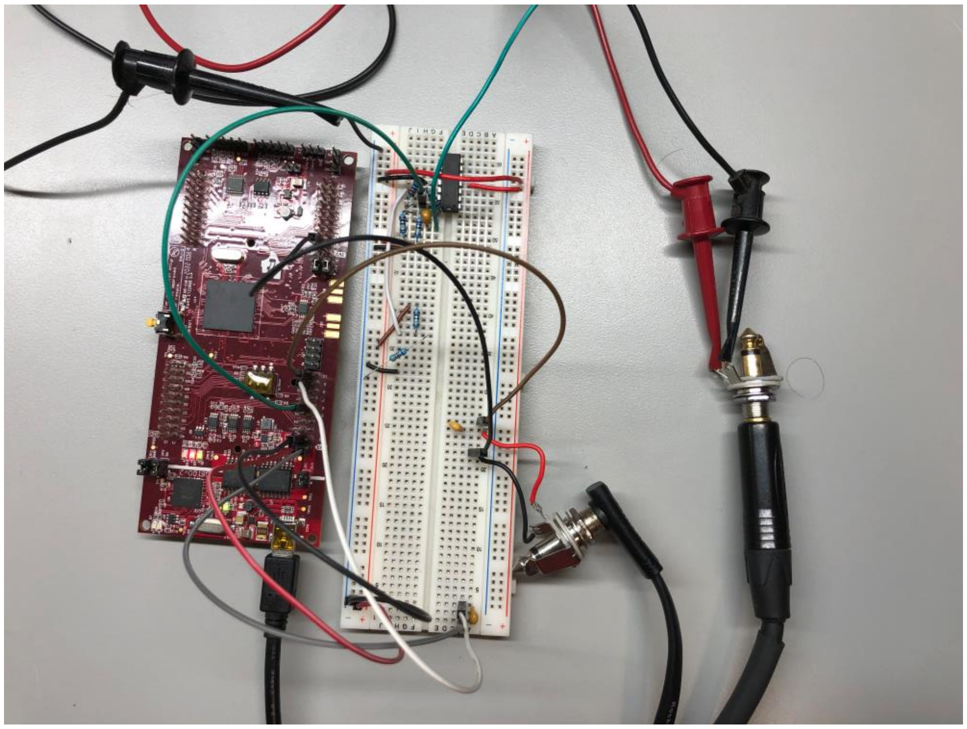
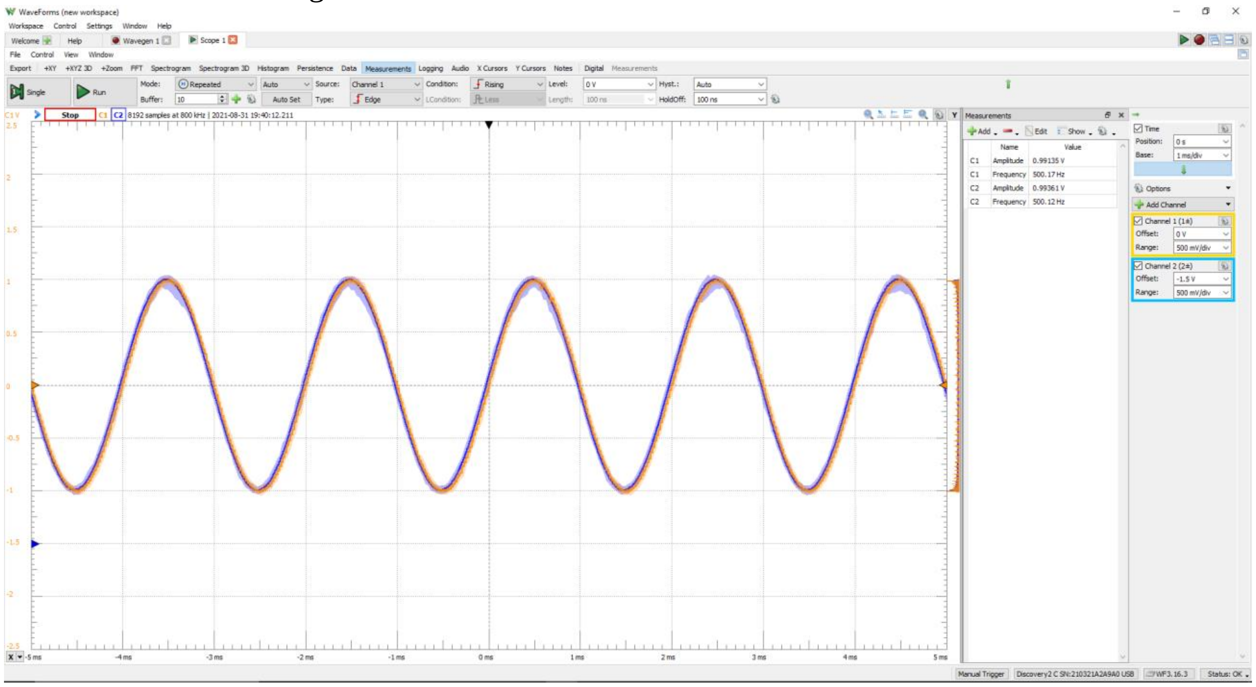


FIGURE IV: DSP Guitar FX Pedal Prototype Hardware Implementation

Chapter 8: Integrated System Tests and Results

Design Verification

For the microprocessor to serve as a platform for digital audio effects, it must accurately read and reconstruct audio signals.



8192 samples at 800 kHz | 2021-08-31 19:40:12.211

Mode: Repeated, Auto; Buffer: 10; Source: Channel 1; Type: Edge; Condition: Rising; Level: 0 V; Hyst.: Auto; HoldOff: 100 ns

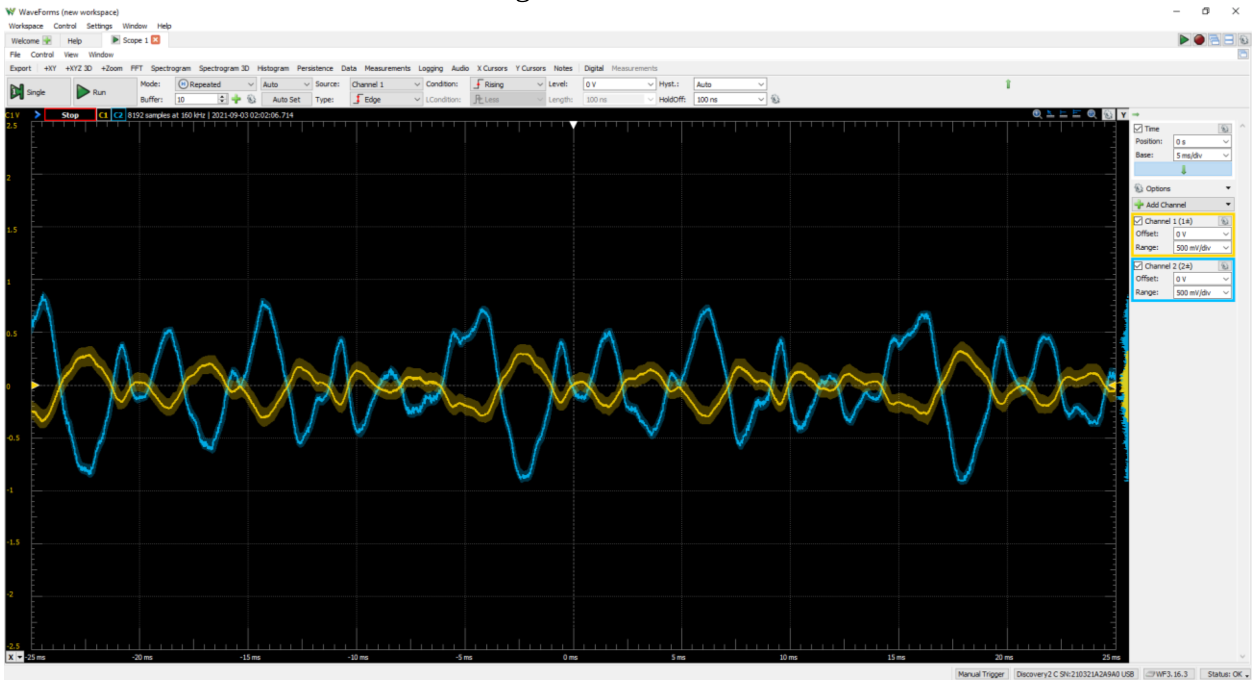
	Name	Value
C1	Amplitude	0.99135 V
C1	Frequency	500.17 Hz
C2	Amplitude	0.99361 V
C2	Frequency	500.12 Hz

Time — Position: 0 s; Base: 1 ms/div

Channel 1 (1±) — Offset: 0 V; Range: 500 mV/div

Channel 2 (2±) — Offset: -1.5 V; Range: 500 mV/div

Figure V: verification of input output waveforms through microprocessor with no effects. Wave generation source



8192 samples at 160 kHz | 2021-09-03 02:02:06.714

Mode: Repeated, Auto; Buffer: 10; Source: Channel 1; Type: Edge; Condition: Rising; Level: 0 V; Hyst.: Auto; HoldOff: 100 ns

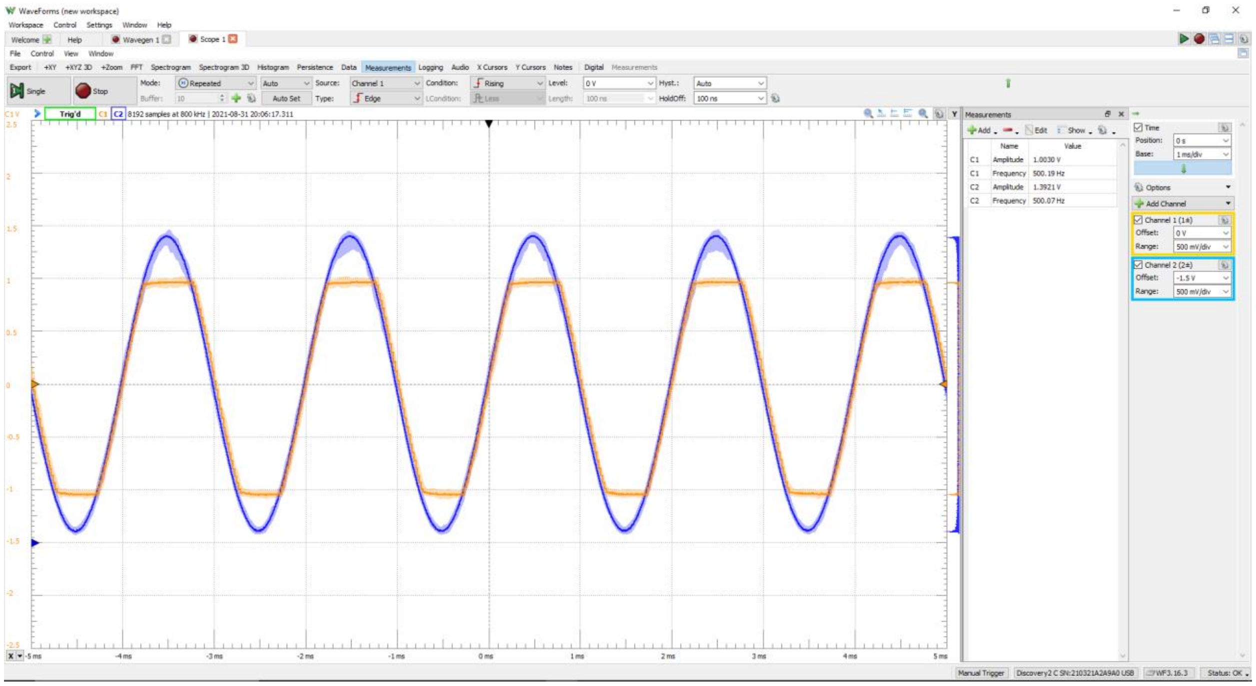
Time — Position: 0 s; Base: 5 ms/div

Channel 1 (1±) — Offset: 0 V; Range: 500 mV/div

Channel 2 (2±) — Offset: 0 V; Range: 500 mV/div

Figure VI: verification of input output waveforms through microprocessor with no effects. Guitar source

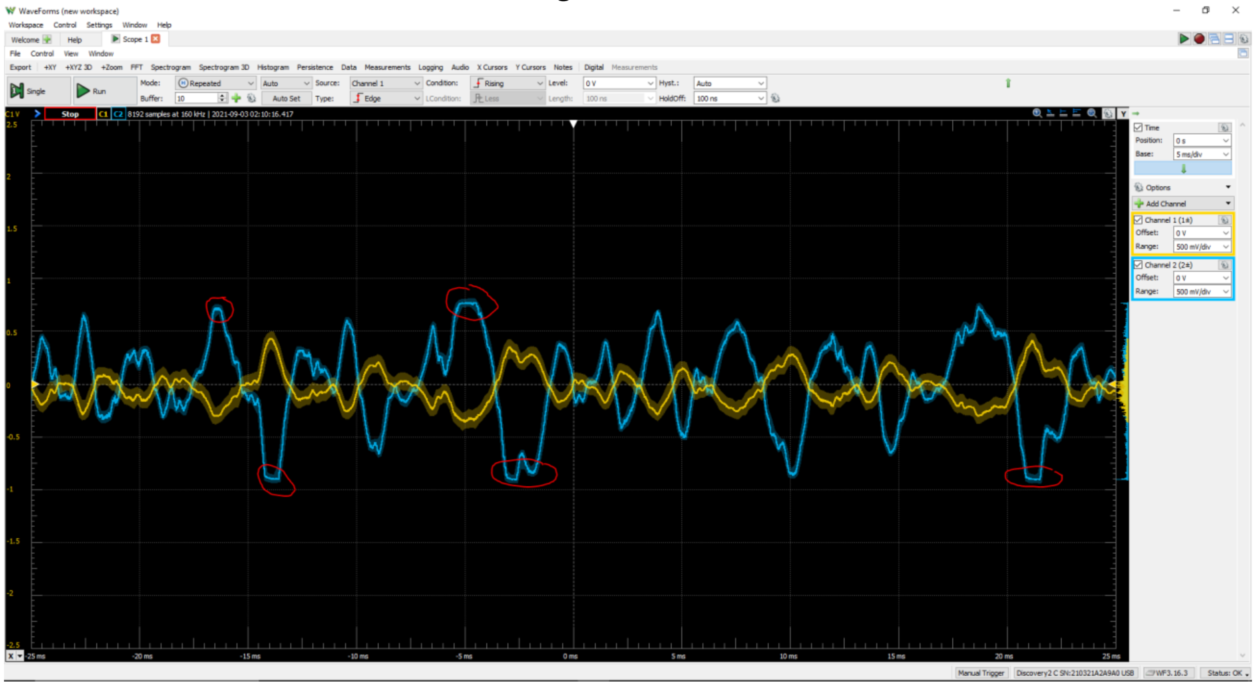
The following figures show the verification of the clipping effect. As shown, the amplitude of the waveforms is limited to the set rail value.



	Name	Value
C1	Amplitude	1.0030 V
C1	Frequency	500.19 Hz
C2	Amplitude	1.3921 V
C2	Frequency	500.07 Hz

Time — Position: 0 s; Base: 1 ms/div
Channel 1 (1±) — Offset: 0 V; Range: 500 mV/div
Channel 2 (2±) — Offset: -1.5 V; Range: 500 mV/div

Figure VII: verification of input output waveforms through microprocessor with clipping effect. Wave generation source



Time — Position: 0 s; Base: 5 ms/div
Channel 1 (1±) — Offset: 0 V; Range: 500 mV/div
Channel 2 (2±) — Offset: 0 V; Range: 500 mV/div

Figure VIII: verification of input output waveforms through microprocessor with clipping effect. Guitar source

In the following figure, the output waveform is shown for the delay effect.

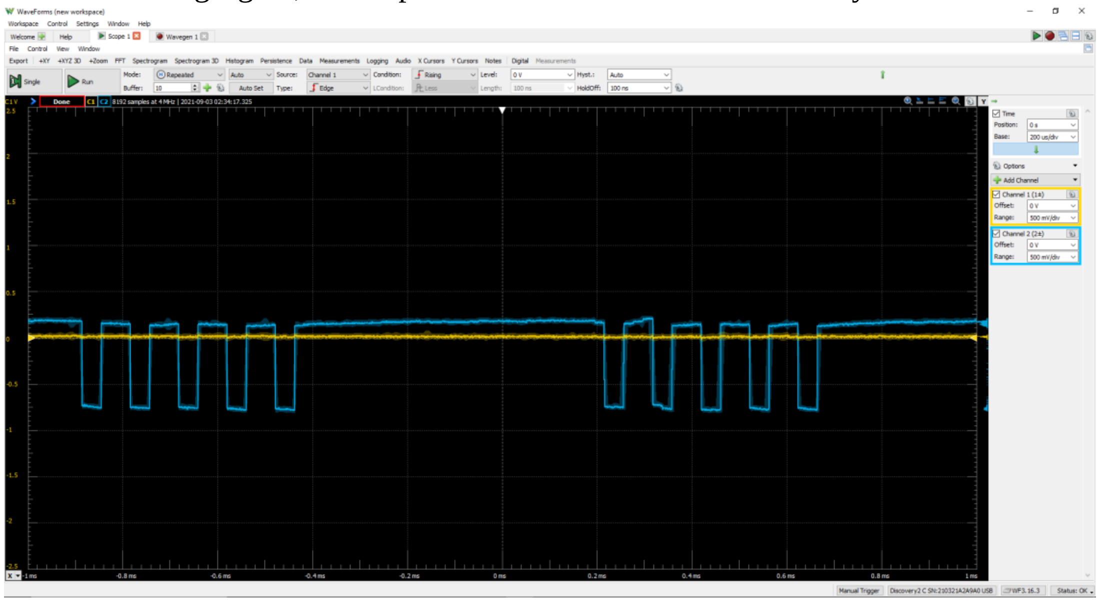


Figure IX: verification of input output waveforms through microprocessor with delay effect. Pulse generation source

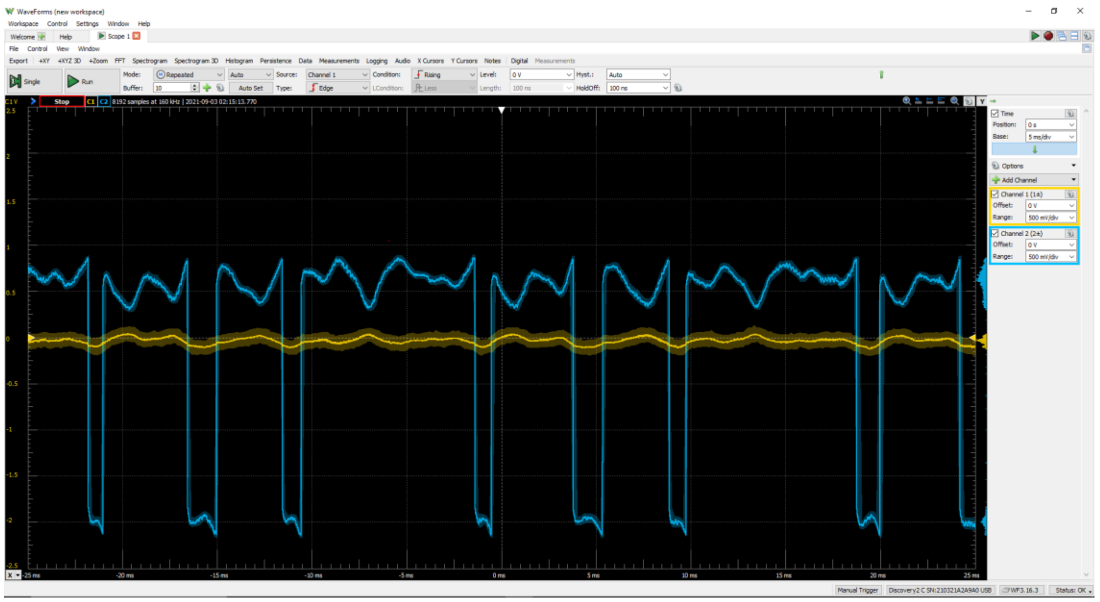


Figure X: verification of input output waveforms through microprocessor with delay effect. Guitar source

Unfortunately, while the delay algorithm works in theory, the limitations in device memory mean only very short delays may be implemented.

Chapter 9: Conclusions

While the proposed specifications of the guitar FX pedal project promised a fully packaged multi-effects unit, the actual project fell short of expectations but did show promise. The project shows working digital processing of real-world signals in form of a guitar input. The current prototype has a working clipping effect and brief delay (referred to as slap-back) and is adjustable through values in the software. The major shortcomings of the project were caused by underestimating the timeline for some tasks. Since I was unfamiliar with the specific microprocessor selected, the setup for the basic digital platform to start implementing effects ran over schedule. The design options made were made to increase production time, and familiar hardware would not perform as needed to serve as a real time digital processor. Therefore, I do not believe that any design adjustments could be made to resolve the main issue in development.

Improvements to be made on the current state of the prototype can be made by following through with development to meet the proposed specifications. Given more development time, I can add more effects to the effects pedal and spend time packaging the hardware in a proper stomp box format. Additions to the current prototype need input knobs and switches to control the effects.

References

[1] R. Ford and C. Coulston, *Design for Electrical and Computer Engineers*, McGraw-Hill, 2007, p. 37 *IEEE Std 1233, 1998 Edition,* p. 4 (10/36), DOI: 10.1109/IEEESTD.1998.

[2] Merriam-Webster, “Musician,” *Merriam-Webster*. [Online]. Available: https://www.merriam-webster.com/dictionary/musician. [Accessed: 02-Sep-2021].

[3] “Silicon shortage expected to continue through 2022,” *Frankenstein Computers, Austintatious IT Support*, 05-Apr-2021. [Online]. Available: https://www.fcnaustin.com/silicon-shortage-expected-to-continue-through-2022/. [Accessed: 02-Sep-2021].

[4] “Universal waste: BATTERIES,” *EPA*. [Online]. Available: https://archive.epa.gov/epawaste/hazard/web/html/batteries.html. [Accessed: 04-Sep-2021].

Appendix A. Senior Project Analysis

Project Title: DSP Guitar FX Pedal

Student's Name: Stephen Rock Student's Signature: Stephen Rock

Advisor's Name: Wayne Pilkington Advisor's Initials: Date: September 3, 2021

• 1. Functional Requirements Summary

The DSP Guitar FX Pedal manipulates an audio signal from a guitar input to create varied and interesting sounds. Therefore, the device must take in an electrical audio signal, process the signal with desired effects, and output the signal to either other guitar pedals or a guitar amplifier. The pedal must have adjustments to the effects and operable by use of a footswitch as standard for guitar pedals.

• 2. Primary Constraints

Table I: Guitar Multi-Effects Pedal Requirements and Specifications outlines the project specifications. The constraints placed on the project stem from operation and current product market expectations. Most pedals on the market use 9 Volt power supplies. The connections for guitar rigs use ¼ inch jacks and must physically fit with other effect pedals on a pedal board. The customer must also be able to turn toggle effects by operation of their foot, due to having both hands occupied by an instrument when using the effects pedal. While some pedals use computer programming to adjust the effects, for ease of use the customer must be able to adjust the effect parameters on the pedal, done through a series of control knobs and switches.

• 3. Economic

The initial cost estimation of the project incudes labor, development equipment, and materials for manufacturing and totals $17,943.29 using PERT cost analysis [1]. The initial costs may either be covered by the development team or investors. The development equipment required only an oscilloscope that totaled $399.00. The materials outlined in Table VI: Bill of Materials Prototype total $44.29. The original estimated development time of the project was based on working 6-8 hours per week on the project for 20 weeks. The actual development time for a complete project would expand past the initial estimate, but the original estimate developed the project to its current state.

• 4. If manufactured on a commercial basis

If manufactured on a commercial basis, say 1,000 units, the costs would reflect Table VII: Bill of Materials Manufacture x1000. The difficulties in manufacturing would include relying on suppliers to for the components and manufactures to produce the circuit boards. However, the current silicon shortage as described in [3] increases costs and decreases availability of silicon-based products such as processors and ICs that are required to manufacture this product. However, using the calculations from [1] the profits of manufacturing and selling 1,000 units would be $37,767, selling each unit at a onetime retail cost of $100 as specified by the project specifications.

• **5. Environmental**
Two main concerns arise with environmental impacts of use and manufacturing of the guitar FX pedal. Firstly, either wall power or a battery powers the device. While wall power does put some load on the power grid, the main concern with battery power relates to the waste produced by using multiple single use batteries. The EPA considers spent batteries as hazardous waste [4] and can therefore cause harm to the environment if improperly disposed of and used in excess. However, the use of rechargeable batteries or wall power reduces the number of disposable batteries used with this product. The second environmental concern relates to the manufacturing of circuit boards and the components used in the product. Circuit boards and silicone chip manufacturing can use chemicals toxic to the environment and other wastes. So, board manufacturing must be selected carefully to ensure that the manufacturing handles waste byproducts safely and minimizes wastes.

• **6. Manufacturability**
As stated before, the silicon shortage currently occurring limits manufacturability of this product as it is dependent on silicon chips. This either makes the manufacturing of the product more expensive or limits the total number of units manufactured. The process for manufacturing the pedal should not prove an issue as the process is typical circuit board manufacturing and soldering, with easy packaging for the electronics.

• **7. Sustainability**
Maintaining the guitar FX pedal should occur if the inevitable defects come from wear and tear of replaceable components such as switches or cable jacks. On the other hand, failure of any on board components such as the processor or signal converters results in difficult to fix issues, and therefore the entire product ceases to function. However, due to the durability and simplicity of the pedal, the pedal should be able to last for at least 10 years. Another impact on the sustainability of this product, the manufacturing uses silicon, copper, and other non-renewable resources. The current silicon shortage as described in [3] calls into question the sustainability of many silicon based products, and lasting production of the guitar FX pedal would be no exception. While the hardware cannot be upgraded postproduction, firmware updates released can upgrade the software and effects included in the original product.

• **8. Ethical**
According to the IEEE Code of Ethics, upholding integrity and responsible behavior in the use, development, and research of technology while treating all persons fairly without discrimination. The use of this product aligns with the IEEE Code of Ethics regarding maintaining the safety of customers, as the device does not operate at any damaging voltage and current and all electronics are packaged to eliminate contact with any unauthorized modification. Improper use of the device in periphery with other amplifiers can cause hearing damage if played at unreasonable volume for the venue for extended periods of time, but that risk is caused by the environment of the product and not the product itself. In reference to ethical code 5 and 6 concerning technical honesty and improvement, only those of which understand the product, which includes the original development team, can modify the product after sale through firmware updates to maintain knowledgeable development of the product based on customer feedback.

• **9. Health and Safety**
Manufacturing condition may cause some health concerns, as electronics manufacturing involves production of wastes materials and toxic chemicals. However, working with responsible manufacturers reduces the health risks involved in manufacturing the guitar pedal. The other risk involved in using the product possibly causes hearing damage. When used in conjunction with the rest of a musician’s rig, exposure to excessive volume from the guitar, effects, and amplifier damages not only the hearing of the user but also the audience of the performance.

• **10. Social and Political**
The design and use of the product do not pose any social or political concerns, however the nature of developing a product for music production meant that the music produces with use of the guitar effects could cause unease due to the sensitive and controversial topics often discussed in music.

• 11. Development

The development of the DSP Guitar FX Pedal included concepts of analog circuit design, microprocessor implementation, programming, signal processing, and project management. The design of the analog circuitry and the verification of the current prototype required knowledge of Op Amps and the use of oscilloscopes. I improved my knowledge and application of Code Composer Studio and the C2000 line of microprocessors that I was only minimally familiar with. Learning the new processor was the most difficult and time-consuming aspect of the project. During the theoretical testing of effect algorithms, I learned the audio design toolbox of the MATLAB software. For the final implementations of the effects, I used C coding and software design methods to write and test the product. Time and cost estimation and time management are new skills I developed throughout this project as complications did arise that modified the optimistic schedule of completing the project.

Appendix B. Parts List and Costs

Cost Estimations: The project cost estimates in Table V, broken down in this section provide several optimistic, pessimistic, and likely costs associated with the project. The PERT analysis from [1], further focuses the cost estimations by providing a likely cost based on best, worst, and likely scenarios. The labor rate used is $35 dollars and hour working 40, 15, and 10 hours a week for worst, realistic, and best-case scenario respectively.

TABLE V

COST ESTIMATE

Item	Worst	Realistic	Best Case
Labor	$28,000	$17,500	$7,000
Analog Discovery 2	$399.00	$399.00	$399.00
Parts as detailed in B.O.M.	$44.29	$44.29	$44.29
Total	$28,443.29	$17,943.29	$7,443.29
PERT Cost Estimate			$17,943.29

Parts List:

TABLE VI

BILL OF MATERIALS PROTOTYPE

Item	**Description**	**Quantity**	**Price per Unit**	**Total Price**
LAUNCHXL-F28379D	C2000 Delfino MCU F28379D LaunchPad™ development kit	1	$33.79	$33.79
Resistor 1 MΩ	1 MΩ Through Hole ¼ W 1%	3	$0.10	$0.30
Resistor 10 kΩ	10 kΩ Through Hole ¼ W 1%	1	$0.10	$0.10
Resistor 27 kΩ	27 kΩ Through Hole ¼ W 1%	1	$0.10	$0.10
Resistor 420 kΩ	420 kΩ Through Hole ¼ W 1%	1	$0.10	$0.10
Capacitor 10 µF	10 µF Ceramic 20%	2	$0.10	$0.20
LMC6484IN	Operational Amplifier CMOS Quad	1	$3.25	$3.25
¼ Inch Audio Jack	Mono ¼ Inch Audio Jack	2	$6.45	$6.45
			Total:	$44.29

TABLE VII
BILL OF MATERIALS MANUFACTURE X1000

Item	**Description**	**Quantity**	**Price per Unit**	**Total Price**
LAUNCHXL-F28379D	C2000 Delfino MCU F28379D LaunchPad™ development kit	1,000	$33.79	$33,790.00
Resistor 1 MΩ	1 MΩ Through Hole ¼ W 1%	3,000	$0.014	$42.00
Resistor 10 kΩ	10 kΩ Through Hole ¼ W 1%	1,000	$0.014	$14.00
Resistor 27 kΩ	27 kΩ Through Hole ¼ W 1%	1,000	$0.014	$14.00
Resistor 420 kΩ	420 kΩ Through Hole ¼ W 1%	1,000	$0.014	$14.00
Capacitor 10 µF	10 µF Ceramic 20%	2,000	$0.014	$28.00
LMC6484IN	Operational Amplifier CMOS Quad	1,000	$1.78	$1,780.00
¼ Inch Audio Jack	Mono ¼ Inch Audio Jack 20 pack	50 (2,000)	$10.99	$549.50
			Total:	$36,231.50

Appendix C. Project Schedule

Project Schedule Overview: This section includes a project planning statement and a table with project deliverables and dates. The Gantt chart further details the tasks and timeline of the project.

Project Planning Statement

This section breaks down deliverable due dates, and work breakdown. The deliverable due dates presented in Table IV cover the general outline of overall important dates related to project delivery and progress. The project Gantt chart in Figure III has a detailed breakdown of project tasks and timeframes.

TABLE IX
GUITAR PEDAL DELIVERABLES

Delivery Date	Deliverable Description
1/22/2021	Design Review
2/26/2021	EE 461 demo
3/12/2021	EE 461 report
8/30/2021	EE 462 demo
9/3/2021	ABET Sr. Project Analysis
9/3/2021	EE 462 Report

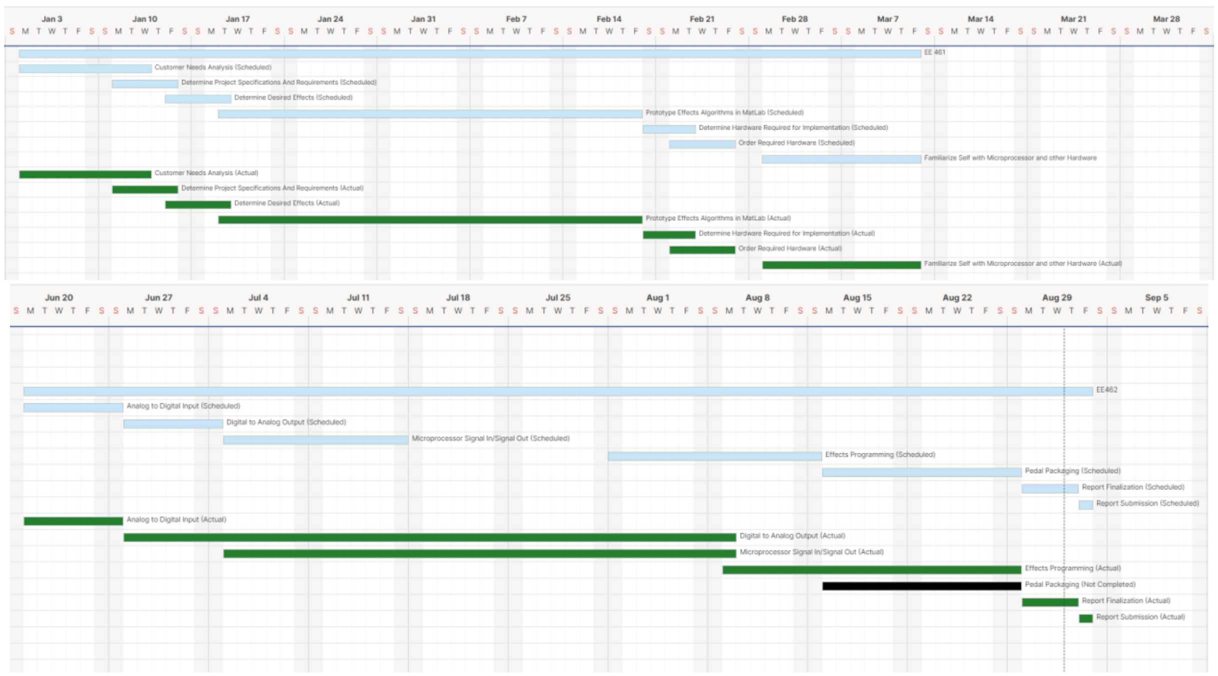


FIGURE XI: DSP GUITAR FX PEDAL GANTT CHART

Appendix D. PC Layout

Schematics:

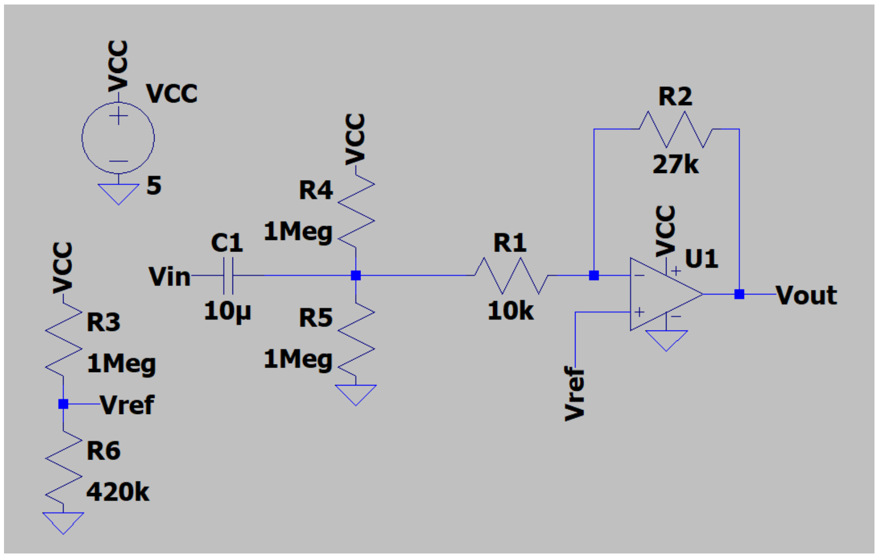


FIGURE XII: INPUT AMPLIFIER SCHEMATIC

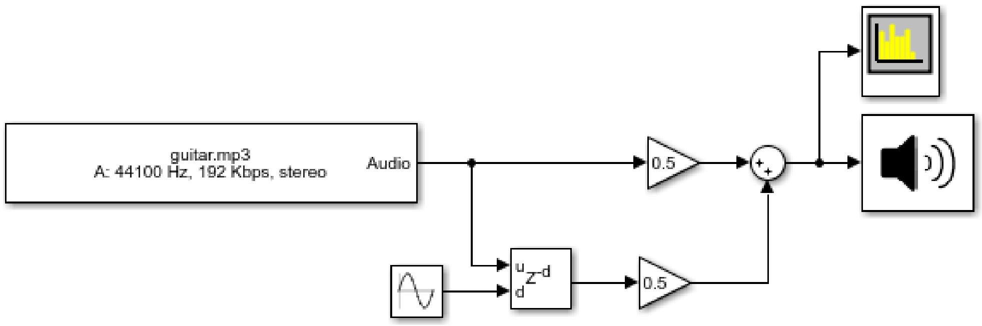
Appendix E. Program Listings

Software Overview: The shown MATLAB code and diagrams are used for experimental proof of concept algorithms to later be implemented in the final build of the pedal.

Tremolo

```
File = dsp.AudioFileReader('guitar.mp3');
Fs = File.SampleRate;
%initialize device writer
Out = audioDeviceWriter('SampleRate', Fs);
%initialize spectrum analyzer
Spectrum = dsp.SpectrumAnalyzer('SampleRate', Fs, 'PlotAsTwoSidedSpectrum',
false, 'FrequencyScale', 'Log');

%parameters
depth = 0.2;
speed = 1;

i = 0.000001;

tic
while toc < 120
    %read block
    x = step(File);

    %tremolo
    ampmod = depth*sin(pi*i*speed).*x;
    y = x + ampmod;
    i = i + 0.000001;
    if i >= 2
        i = 0.000001;
    end

    %write block to output
    step(Out,y);

    %plot
    figure(1)
    plot(x)
    drawnow
    %visualize spectrum
    step(Spectrum, [x(:,1),y(:,1)])
end
```

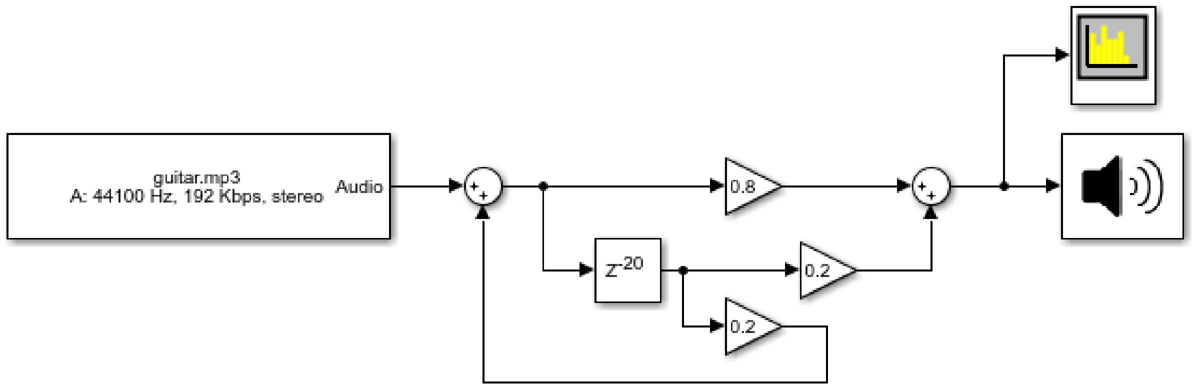
Flanger



```
File = dsp.AudioFileReader('guitar.mp3');
Fs = File.SampleRate;
%initialize device writer
Out = audioDeviceWriter('SampleRate', Fs,'SupportVariableSizeInput',true);
%initialize spectrum analyzer
Spectrum = dsp.SpectrumAnalyzer('SampleRate', Fs, 'PlotAsTwoSidedSpectrum',
false, 'FrequencyScale', 'Log');

%parameters
depth = 0.015; %range of variable time delay
speed = 100; %

i = 0.00001;

tic
while toc < 120
    %read block
    x = step(File);

    %Flanger
    delay = depth*sin(pi*speed*i);
    i = i + 0.000000001;
    if i >= 2
        i = 0.000000001;
    end
    %y[n] = x[n] + decay*x[n-delay*Fs]
    Dk=round(delay*44100);
    echo_filter_hn=[1 zeros(1,Dk(end)-1)];
    echo_filter_hn(Dk+1)=1;

    xnL=length(x);
    hnL=length(echo_filter_hn);
%     calculate optimal length M
    M=xnL+hnL-1;
%     for speed force to 2^N, where N is returned by nextpow2
    Mpow2=2^nextpow2(M);
%     fft both input and unit sample response
    FTX=fft(x(:,1).',Mpow2);
    Flang=fft(echo_filter_hn,Mpow2);
```

```
%     linear-convolution in time is multiplication in DFT:
    FTY=FTX.*Flang;
%     inverse transform with same N-point for speed
    ynPad=abs(ifft(FTY,Mpow2));
%     splice out the extra zeros created by speed padding
    y=ynPad(1:M).';

    %write block to output
    step(Out,y);

    %plot
    figure(1)
    plot(x)
    drawnow
    %visualize spectrum
    %step(Spectrum, [x(:,1),y(:,1)])
end
```

Delay



Software Overview: The shown C code written in Code Composer Studio is prototype implementation on the C2000 TMS320F28379D Launchpad.

```
//
// Included Files
//
#include "F28x_Project.h"
#include "CLAmath.h"
#include <stdio.h>

//
// Defines
//
#define RESULTS_BUFFER_SIZE 1 //256
#define DLOG_SIZE           1 //256
#define REFERENCE_VDAC      0
#define REFERENCE_VREF      1
#define DACA                1
#define DACB                2
#define DACC                3
#define REFERENCE           REFERENCE_VREF
#define CPUFREQ_MHZ         200
#define DAC_NUM             DACA

//
// ADC Globals
//
Uint16 AdcaResults[RESULTS_BUFFER_SIZE];
//Uint16 resultsIndex;
//volatile Uint16 bufferFull;

//
// DAC Globals
//
Uint16 DataLog[DLOG_SIZE];
#pragma DATA_SECTION(DataLog, "DLOG");
volatile struct DAC_REGS* DAC_PTR[4] = {0x0,&DacaRegs,&DacbRegs,&DaccRegs};
Uint32 samplingFreq_hz = 98000;

Uint16 val_out = 0;
Uint16 ndx = 0;
float freqResolution_hz = 0;
float cpuPeriod_us = 0;
Uint32 interruptCycles = 0;
float interruptDuration_us = 0;
float samplingPeriod_us = 0;

float dc_offset = 0x0810;

bool clipping = false;
float clip_max = 0x0D70;
float clip_min = 0x02C8;
/*
```

```
bool tremelo = false;
Uint32 speed = 20; // Frequency of Trem in Hz
float depth = 0x0001;
Uint32 sine_size = 1; // Number of samples @fsample for sine wave of frquency speed
float sine_lookup[100];
Uint32 pointer = 0;
float temp_Angle;
*/
bool delay = true;
float delay_time = 0.01; //seconds
Uint32 delay_num_samples = 1; //delay_time * samplingFreq_hz;
Uint16 delay_sample_buffer[980];//Uint16 delay_sample_buffer[delay_num_samples];
//buffer for samples
Uint32 pointer2 = 0;

//
// Function Prototypes
//
static inline void dlog(Uint16 value);
void configureDAC(Uint16 dac_num);
void ConfigureADC(void);
//extern void AdcSetMode(Uint16 adc, Uint16 resolution, Uint16 signalmode);
interrupt void cpu_timer0_isr(void); // ADC+DAC
//void Update_Tremelo_Lookup_Table(Uint32 speed, float depth, Uint32
samplingFreq_hz);
//float CLAsin(float fAngleRad);

//
// Main
//
void main(void)
{
//
// Initialize System Control:
// PLL, WatchDog, enable Peripheral Clocks
// This example function is found in the F2837xD_SysCtrl.c file.
//
    InitSysCtrl();

//
// Disable CPU interrupts
//
    DINT;

//
// Initialize the PIE control registers to their default state.
// The default state is all PIE interrupts disabled and flags are cleared.
// This function is found in the F2837xD_PieCtrl.c file.
//
    InitPieCtrl();

//
// Clear all interrupts and initialize PIE vector table:
```

```
//
    IER = 0x0000;
    IFR = 0x0000;
    InitPieVectTable();

//
// Map Cpu Timer0 and Timer1 interrupt function to the PIE vector table
//
    EALLOW;
    PieVectTable.TIMER0_INT = &cpu_timer0_isr;
    EDIS;

//
// Initialize variables
//
    cpuPeriod_us = (1.0/CPUFREQ_MHZ);
    samplingPeriod_us = (1000000.0/samplingFreq_hz);

//
// Initialize datalog
//
    for(ndx=0; ndx<DLOG_SIZE; ndx++)
    {
        DataLog[ndx] = 0;
    }
    ndx = 0;

//
// Configure DAC
//
    configureDAC(DAC_NUM);

//
// Configure ADC
//
    ConfigureADC();

//
// Initialize Cpu Timers
//
    InitCpuTimers();

//
// Configure Cpu Timer0 to interrupt at specified sampling frequency
//
    ConfigCpuTimer(&CpuTimer0, CPUFREQ_MHZ, samplingPeriod_us);

//
// Initialize Sine lookup table for Tremelo Effect
//
    //Update_Tremelo_Lookup_Table(speed, depth, samplingFreq_hz);
```

```
//
// Setup number of samples needed for delay
//
    delay_num_samples = delay_time * samplingFreq_hz;
    //delay_sample_buffer[delay_num_samples] = {0};

//
// Start Cpu Timer0
//
    CpuTimer0Regs.TCR.all = 0x4000;

//
// Enable interrupt
//
    IER |= M_INT1;
    PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
    EINT;
    ERTM;

    while(1)
    {
    }
}

//
// dlog - Circular DataLog. DataLog[0] contains the next index to
//        be overwritten
//
static inline void dlog(Uint16 value)
{
    DataLog[ndx] = value;
    if(++ndx == DLOG_SIZE)
    {
        ndx = 0;
    }
    DataLog[0] = ndx;
}

//
// Update Tremelo Sine Lookup Table
//
/*void Update_Tremelo_Lookup_Table(Uint32 speed, float depth, Uint32
samplingFreq_hz){
    sine_size = samplingFreq_hz/speed;
    for(pointer = 0; pointer < sine_size; pointer = pointer + 1 ){
        temp_Angle = (2*3.14*pointer)/sine_size;
        sine_lookup[pointer] = CLAsin(temp_Angle);
    }
}*/

//
// ConfigureADC - Write ADC configurations and power up the ADC for both
//                ADC A and ADC B
```

```
//
void ConfigureADC(void)
{
    EALLOW;

    //
    //write configurations
    //
    //AdcSetMode(ADC_ADCA, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);
    AdcaRegs.ADCSOC0CTL.bit.CHSEL = 3; //SOC0 will convert ADCINA3
    AdcaRegs.ADCSOC0CTL.bit.ACQPS = 24; //SOC0 will use sample duration of 24
SYSCLK cycles
    AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 1; //SOC0 will begin conversion on CPU1 Timer
0

    //
    //power up the ADC
    //
    AdcaRegs.ADCCTL1.bit.ADCPWDNZ = 1;

    //
    //delay for 1ms to allow ADC time to power up
    //
    DELAY_US(1000);

    EDIS;
}

//
// configureDAC - Enable and configure the requested DAC module
//
void configureDAC(Uint16 dac_num)
{
    EALLOW;

    DAC_PTR[dac_num]->DACCTL.bit.DACREFSEL = REFERENCE;
    DAC_PTR[dac_num]->DACOUTEN.bit.DACOUTEN = 1;
    DAC_PTR[dac_num]->DACVALS.all = 0;

    DELAY_US(10); // Delay for buffered DAC to power up

    EDIS;
}

//
// cpu_timer0_isr - Timer ISR that writes the sine value to DAC, log the sine
//                  value, compute the next sine value, and calculate interrupt
//                  duration
//
interrupt void cpu_timer0_isr(void)
{
    //
    // Start Cpu Timer1 to indicate begin of interrupt
    //
    CpuTimer1Regs.TCR.all = 0x0000;

    //
    // Take in ADC input value(s)
    //
    AdcaResults[0] = AdcaResultRegs.ADCRESULT0;

    //
    // PROCESSING
    //
    val_out = AdcaResults[0];
    // Clipping effect
    if (clipping == true){
        //val_out = val_out - 2081; // 2081 Decimal 1.5 V offset
        if (val_out > clip_max){
            val_out = clip_max;
        }
        if (val_out < clip_min){
            val_out = clip_min;
        }
    //val_out = val_out + 2081;
    }

    // Tremelo effect
    /*if (tremelo == true){
        val_out = val_out * sine_lookup[pointer];
        pointer = pointer + 1;
        if (pointer == sine_size){
            pointer = 0;
        }
    }*/

    // delay effect
    //float delay_time = 0.5; //seconds
    //Uint16 delay_num_samples = 1; //delay_time * samplingFreq_hz;
    //Uint16 delay_sample_buffer[delay_num_samples]; //buffer for samples
    if (delay == true){
        //fill up delay buffer
        //delay_num_samples = delay_time * samplingFreq_hz;

        //chaser pointers in delay buffer for delay
        if (pointer2 > delay_num_samples){
            pointer2 = 0;
        }
        delay_sample_buffer[pointer2] = val_out;
        if (pointer2 == 0){
            val_out = delay_sample_buffer[pointer2] +
delay_sample_buffer[delay_num_samples - 1];
        }
        else{
            val_out = delay_sample_buffer[pointer2] + delay_sample_buffer[pointer2
- 1];
```

```
        }
        pointer2 = pointer2 + 1;
    }

    //
    // Write current output value to buffered DAC
    //
    DAC_PTR[DAC_NUM]->DACVALS.all = val_out;

    //
    // Log current DAC value
    //
    dlog(val_out);

    //
    // Acknowledge this interrupt to receive more interrupts from group 1
    //
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

    //
    // Stop Cpu Timer1 to indicate end of interrupt
    //
    CpuTimer1Regs.TCR.all = 0x0010;

    //
    // Calculate interrupt duration in cycles
    //
    interruptCycles = 0xFFFFFFFFUL - CpuTimer1Regs.TIM.all;

    //
    // Calculate interrupt duration in micro seconds
    //
    interruptDuration_us = cpuPeriod_us * interruptCycles;

    //
    // Reload Cpu Timer1
    //
    CpuTimer1Regs.TCR.all = 0x0030;
}

//
// End of file
//
```