Analog and Digital Effects Processing Technology (ADEPT)

Diego Conterno, Tyler Michaud, Alejandro Porcar, Dylan Walter

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — Unlike the regular guitar pedals in the market, the ADEPT system aims to bring an affordable solution to develop and test out a multitude of effects. In an effort to bring to life the use of DSP in the guitar effects world, our goal is to provide the user with a platform for implementing any effect they can imagine. Our system can capture, process, store and manipulate the signal from a guitar and then output the modified signal.**

***Index Terms* — Filters, Codecs, Digital Audio Player.**

I. Introduction

As engineering students, as well as musicians, we find that a senior design project based around audio engineering, music technology, and sound design would be very rewarding and informative. It would be the perfect opportunity to apply the engineering fundamentals we have learned into something we are passionate about. Closing the gap between technology and music, we aim to provide new alternatives for musicians who want to stand out and shape their sound in a unique way. We are attempting to provide a digital multi-effects solution that will have a creative impact on musicians with an interest in electrical and computer engineering.

Therefore, we designed a musical instrument effects processing unit. This simple-to-use effects pedal is able to take the analog input from a guitar or other musical instrument (via a ¼" instrument cable), convert that analog signal into a digital signal that can be manipulated, and then be converted back into an analog signal.

II. Project Description

The digital signal processing (DSP) is done by the microcontroller.

The CODEC in combination with the MCU allows the transition from the frequency domain to the discrete domain, in other words, converting an analog signal to a digital bitstream, which is processed on a sample-by-sample basis and then sent back to the CODEC in order to be converted to an audible analog signal. Since most guitar pedals that are commercially available run off of 9V – 12V (DC), the power requirements would be very simple to integrate.

III. Hardware Components/Specifications

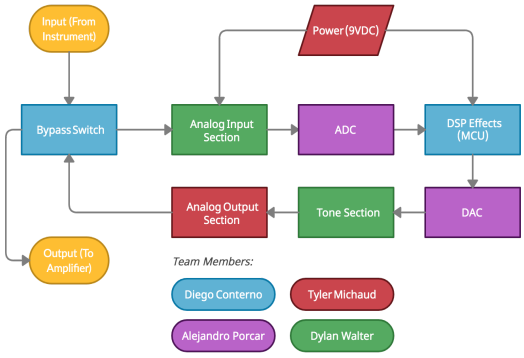


Figure 1: Block Diagram

In this section we will outline the hardware components of our design. This will include the input and output buffers of the circuit, as well as the tone and volume controls, digital hardware, user peripherals, and the power distribution layout of the system.

A. Input/Output Buffers

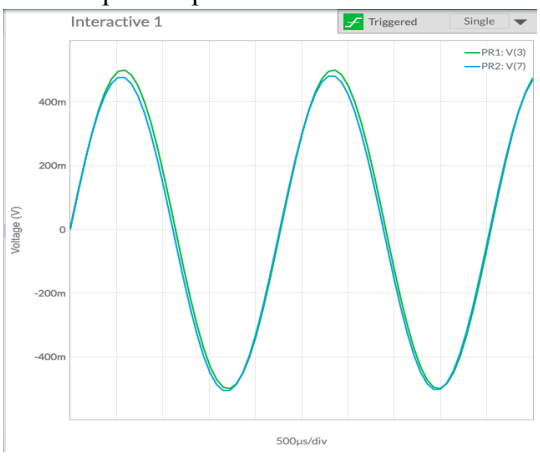


Figure 2: Input Buffer Simulation

A buffer is an active electronic circuit that can provide a change in electrical impedance, or resistance. When it comes to audio circuits, impedance is directly related to sound quality. The longer the signal path (distance between the instrument and the amplifier), the higher the impedance. The higher the impedance of the circuit, the darker the tonal quality of the signal will become. High frequencies will get removed from the signal, resulting in muddiness and a lack of clarity.

The reason for this necessary change in impedance in our circuit is due to the effect that impedance has on a guitar's signal. If a musician is running his/her guitar through a long signal path of effects pedals, or even just a long instrument cable (both of which increase in capacitance the longer they are), the tone of the instrument will degrade significantly, and the resilience and clarity of the frequency spectrum will become compromised.

We implemented a common collector/emitter follower buffer configuration for both the analog input and output stages of our circuit. We designed both buffers to have unity gain of one, with our input buffer having a high input impedance and our output buffer having a low output impedance. In order to mitigate sound quality issues, we must ensure that the input and output impedances of our buffers are properly calculated such that there is no signal loss or tonal compromise. The input impedance of our guitar pedal at the input buffer should be anywhere between 470kΩ to 1MΩ. Any less than this range would yield a muddy or muffled sound, and any more would have the opposing effect of a bright and thin sound that is harsh to the ears. This impedance range that comes directly from the guitar's pickups is very high, and will need to be brought down to a lower impedance level to prepare to enter into the CODEC.

The maximum value of the 2N2222A transistor's DC current gain (β) is 300. We can use this value to assist in calculating the input and output impedances of our buffer. Since we know that we want our input impedance to be high (between 470kΩ to 1MΩ), we can use this range as a reference when plugging in resistor values during our small signal AC analysis. The value of rπ is negligible in our analysis, since it is in parallel with resistor R2 in our circuit, and has an incredibly small value relative to the value of R2. The base of the transistor must be biased to 4.5V in order for it to turn on and enter into the forward active region. The coupling capacitors in our buffer are placed such that the AC signal coming from the guitar and the DC signal coming from the 4.5V bias voltage do not mix.

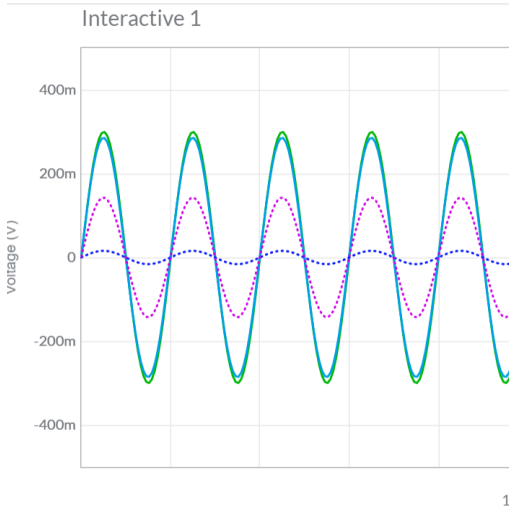


Figure 3: Output Buffer Simulation

The design of the output buffer is very similar to the development of our input buffer. We created a hybrid-pi model to represent the small signal circuit of the output buffer. Since we are finding the equivalent impedance from the small signal circuit we can eliminate the current source from the model. This leaves the current source as an open circuit. Also take note that rπ is much smaller than load at the output. Hence we can eliminate rπ from the calculations. The nominal input impedance of an input buffer is between 450kΩ and 750kΩ. The nominal output impedance of an output buffer is between 400Ω and 500Ω.

For our input buffer, we calculated 750kΩ input impedance and measured a value of 682kΩ with an impedance analyzer. For our output buffer, we calculated 475Ω output impedance and measured a value of 419Ω with an impedance

analyzer. Although the calculated values of the input and output impedance of our buffers deviated from measured values, they still are within the nominal range.

B. Power Section

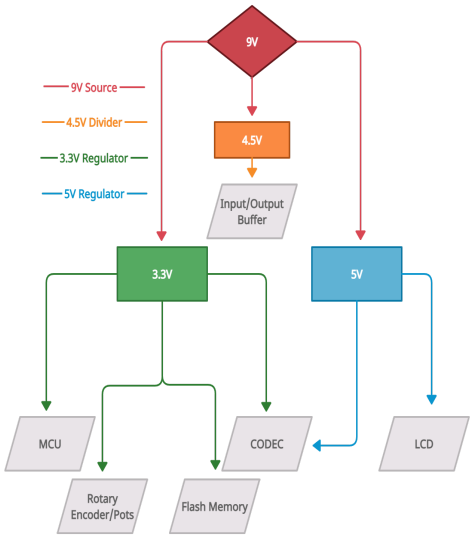


Figure 4: Power Distribution Layout

We chose to power our guitar pedal with 9V since it is a standard voltage to power guitar pedals on the market. We implemented linear voltage regulation to power our devices. The advantages of linear regulators are that they have a simple design, low noise due to the absence of switching, few external parts, and low cost. We used a 3.3V step down linear regulator to power the STM32 MCU, PCM3060 CODEC, rotary encoder, pots, and flash memory. We also implemented a 5V step down linear regulator to power the LCD and the PCM3060 CODEC (for I2C configuration).

We implemented a single ground plane for both digital and analog ground since our signal frequencies are low enough to not be disturbed from cross talk or digital ground noise since digital ground noise tends to become an issue at 5Ghz. We chose our 9V master power source to be an isolated 500mA DC power supply with a standard wall plug in and a center positive 2.1 x 5.5 mm output barrel plug. We chose this method to simplify our design without having to rectify and isolate the AC 115 VAC – 120 VAC, 60 Hz voltage from a standard wall plug-in. This also saves space on the PCB.

We selected an AMS1117-3.3V DC 4.75V-12V to 3.3V linear voltage regulator and a LM7805 step down 5V linear voltage regulator. The 3.3V regulator has an output current of 1A and an operational input voltage of 4.75V to 15V. The 5V regulator has an output current of 1.5A and an operational input voltage of 7V and 25V. Given that the total current draw will be under 1A and the input voltage of 9V is well within the margins of operational input voltage of each regulator, the regulators will be able to produce their specified output voltages and produce plenty of current without the regulators dropping out. Overall, we chose these regulators for their simplicity of design, low cost, and low noise.

C. Tone and Volume Control

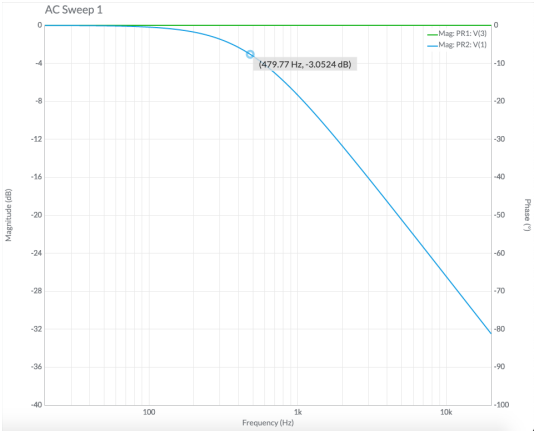


Figure 5: Tone Control Frequency Response

On most guitar pedal designs, there are some utility controls available onboard to adjust some analog parameters of the circuit: volume and tone. Both of these parameters can be controlled with a simple variable resistor (potentiometer). The tone control is used in order to make the sound of the signal brighter or darker, and the volume control is used to adjust the overall amplitude level of the signal leaving the circuit.

The section of our circuit that will come directly after the digital effects processing stage of our effects pedal is the tone section. This section will be responsible for shaping the overall tone of the effect. A tone control on any piece of audio gear is generally defined as a simplified method of equalization that can directly affect

the amplitude of low, middle, and high frequencies in the human hearing range.

For our tone control design, we implemented the "Bluesbreaker" style of tone control, which is a simple First Order Low Pass RC Filter. We designed this filter with a cutoff frequency of 500Hz. As the user turns the knob to the left, the tone gets darker (cutoff frequency decreasing) and when the user turns the knob to the right, the tone gets brighter (cutoff frequency increasing).

For our volume control, we simply implemented a potentiometer at the very end of our circuit, which functions by bleeding our signal to ground, reducing the amplitude of the output signal.

D. Peripherals

In order to provide a way for the user to interact with the guitar effects pedal, we implemented a few peripherals to control all functionalities and that create an intuitive experience for the user. We included a 16x2 LCD screen to display the parameter values and current effect, and a rotary encoder to allow navigation between different operation modes and effects. The user can also interact with several potentiometers that act as dynamic controls that can change certain parameters within each selected effect. We have also implemented a mechanical 3PDT (3-Pole, Double Throw) latching footswitch to activate the effect circuit, as well as a momentary SPST (Single Pole, Single Throw) footswitch that is used to interface with the looper mode and provide tap-tempo functionality to the timed effect parameters. In order to provide even more user feedback, there are two LEDs onboard the ADEPT that display the status of several parameters (on/off and tap tempo/looper state). For timer testing purposes, a "heartbeat LED" is also present on our PCB.

E. MCU (STM32F446RC)

The microcontroller chosen was the STM32F446RC. This device is responsible for performing the Digital Signal Processing to produce the desired effects, as well as driving the several other peripherals in this design. The package chosen was that of a 64 pin layout, although is not the largest one, the amount of peripherals we have is few and we didn't use all the pins. The STM32 is running at a maximum clock speed of 180MHz, and contains a flash memory capacity of 256kB and SRAM of 128kB. We took advantage of some of the included features such as the general purpose DMA1 and DMA2, which proved to be essential for memory management and data flow of the audio buffers. We also decided to equip the MCU with an external clock of 8 MHz for stability. We were able to program the chip through an external SWD/JTAG debugger linking device in accordance with the STM32CubeIDE to assign pins and flash the code to memory.

F. CODEC (PCM3060)

The PCM3060 by Texas Instruments is a device that contains both an ADC and a DAC converter in one chip. It is necessary for digitizing the guitar signal for it to be processed by the MCU, as well as reverting the modified signal back to analog to be reproduced as sound. This IC can achieve a resolution of up to 24-Bits at a sampling rate of 48KHz. I2C is used for mode control purposes and I2S for transferring audio data. To function properly this chip needs to be clocked via an external oscillator crystal running at 12.288MHz. The analog output lines of the PCM3060 are directed to the analog output buffer before going to an amplifier.

G. Flash Memory (W25Q128JVSIQ TR)

In order to have long enough sample storage for an additional looping function, we will require an external flash memory. The internal 256 KB flash memory embedded on the MCU won't be enough to store long samples. For example, if we are recording audio at a sampling frequency of 48 kHz with a 24 bit resolution, then we can calculate the required amount of memory, which will be equal to 44,800 samples/second * 24 bits/sample = 1,075,200 bits (134.4 kB). The memory cost for recording a second is already too much for the internal flash memory of the MCU, and the usual guitar loop is between 1 - 5 seconds. However, no signal processing was envisioned for the recording of a loop. We would only record and playback a clean guitar signal.

IV. Software System Concept

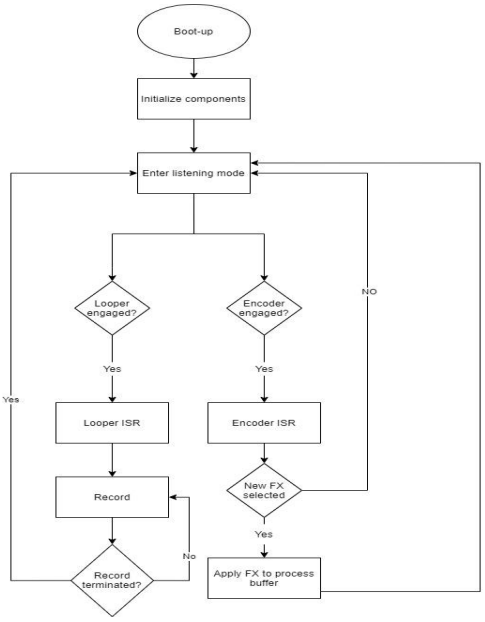


Figure 6: System Flowchart

The system flowchart shown above is a depiction of how the firmware reacts based on the decisions done by the user. As you can see, the system is non-conclusive in nature, meaning that since the system is started, then it won't stop until the whole system is shut down.

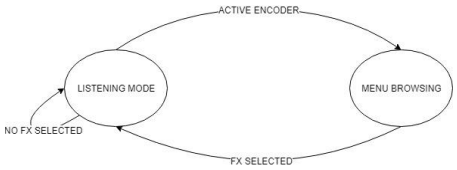


Figure 7: System State Machine Diagram

The two-state machine diagram we see above shows the non-conclusive nature of the system. The system is mainly driven in the state of "listening mode" which refers to the state in which we are storing incoming data from the ADC component of the CODEC, processing it, and then transmitting it to the DAC component of the CODEC. The system will remain in a constant listening mode unless we decide to press the reset button or power it off. Menu browsing using the rotary encoder will not interrupt the audio processing. When one of the effects is selected, then the system immediately applies the corresponding DSP function to the values stored in the audio buffer. The "audio buffer" is a combination of two array buffers of type uint16_t.

V. Software Details

A. I/O Buffers & DMA

Both of these buffers are used by the DMA1 to store and transmit the data using a circular algorithm. Which allows us to have a constant flow of data into our buffers and automatically reloaded.

The DMA circular buffer is configured by HAL automatically, so there's no required code implementation, however for audio applications it is recommended to also use a Ping-Pong buffer algorithm with the DMA's circular buffer, this speeds up the process of applying the DSP function to the values received. The implementation of the Ping-Pong algorithm was done on a sample-by-sample basis, where the data is introduced in pairs (L/R), a complete sample constitutes two left samples and two right samples, alternated. Therefore, a full single sample required 4 cells in a uint8_t array. The algorithm requires us to have a size of 2N, where N is 4.

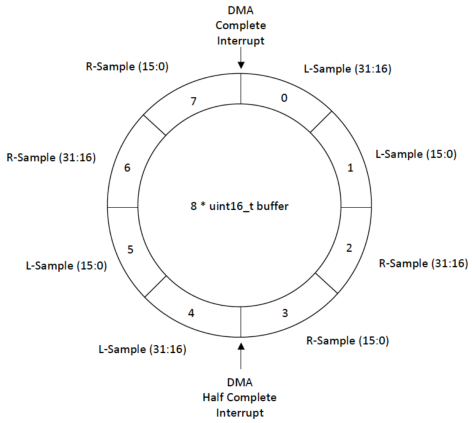


Figure 8: Ping-Pong Buffer

The reason we want to have the two halves is so that the DMA can safely access one of the halves while we apply the DSP function to the other. In the time that the DMA takes to fill up one half of the receive buffer with incoming samples from the I2S, we are processing the other half of the receive buffer. The DMA is

simultaneously transmitting out the most recently processed block of sample. In our case, the block would be of size 1 since we are doing sample-by-sample basis. With the use of the DMA's half-full and full buffer interrupts we let the DMA handle transferring to and from the CODEC whilst the processor handles the signal processing. This is a highly efficient audio stream algorithm that helps the processor devotes most of its resources to only audio processing while the memory management is handled by the DMA.

B. STM32 HAL

The Hardware Abstraction Layer is a compilation of drivers created by STM with the intention of increasing development speed and productivity across all of their STM32 line of chips. The HAL drivers offer a series of APIs which simplify the setup and usage of most generic peripherals included in the STM32. It proved especially valuable when dealing with using I2S along with DMA since it was a concept very new to us and allowed us to get to development quicker. However, a big downside of using HAL is its difficulty to debug once something went wrong. Having to dive deep into nested libraries with high-level functions and routines to find out what could be simple bugs, became an issue.

VI. Effects & Algorithms

A. Delay

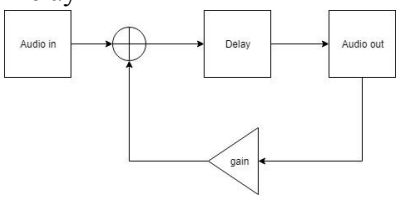


Figure 10: Feedback Comb Filter

The implementation we utilized for our delay simulates that of a feedback comb filter. This particular comb filter is an IIR ("recursive") digital filter since there's *feedback* from the delayed output to the input. The result is a sequence of "echoes" exponentially decaying.

$$y[n] = b_0 x[n] - a_M y[n - M].$$

Figure 11: Difference equation

For stability we want $|a_M| <= 1$, otherwise the result would be an infinitely increasing series of echoes, where each one will be louder than the previous one.

B. Distortion

In order to reproduce the effect of distortion, the original signal must be hard clipped at the top and bottom of the wave, as shown in the figure above. To achieve this digitally we implemented a program where we are able to amplify the incoming signal, as well as specify the thresholds of +Vcc and -Vcc such that the amplified signal will be distorted/clipped at the top and bottom of the signal's waveform.

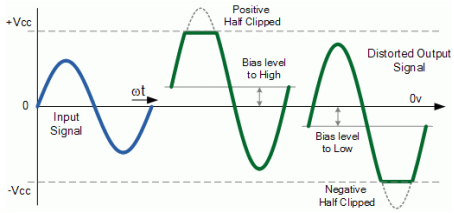


Figure 12: Distortion/Clipping Diagram

The values by which the signal was modified are determined by the parameters set by the user through the parameter potentiometers. These parameters are Amplitude, Threshold, and Mix/Blend.

C. Allpass/"Pipe"

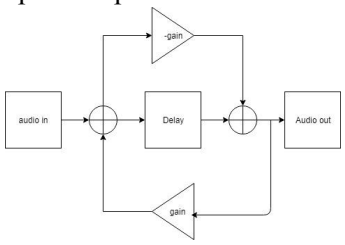


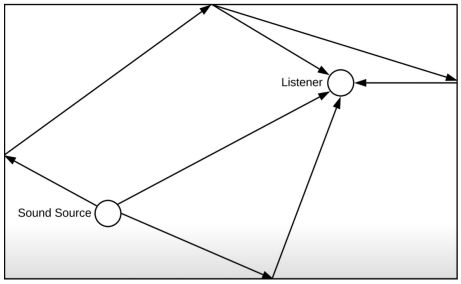
Figure 11: Allpass Filter Diagram

The allpass filter shown above is analogous to the all-pass filter from analog filters. As the name implies it allows pass of all frequencies, in other words the amplitude response of an all pass filter equals to 1 for all the frequencies, whilst the phase response (determines delay vs. frequency) can be arbitrary. For our implementation we have changed the gain parameter to 0.7, this results in a sound similar

to if you played a signal through a pipe, hence the name “pipe”.

D. Reverb

A reverb filter tries to emulate the physical model of sound in a room, where there’s multiple different reflections of the sound coming from the source.



The implementation used for the reverb filter follows Schroeder reverberation algorithm. Which is a popular reverb algorithm published by Schroeder for the Audio Engineering Society.

The way it is implemented is using 4 comb filters in parallel with 3 all pass filters. There’s also a feedforward that allows us to use the parameter of dry/wet, allowing the user to determine how much of the clean signal will be added to the output.

E. Compressor

The goal of a compressor is to gradually reduce the overall gain of an input signal if it is passed a certain threshold. The compressor has three main parameters: Attack, release, and hold. If the input signal jumps to a higher level above the threshold, then the attack phase is initiated; attack means that the audio input signal level is reduced step-by-step over time until the final gain reduction is complete. The attack parameter is set to a specific time in milliseconds. Once the signal level is below the threshold then the compressor recognizes this and waits for the hold time to set the signal back to the 0 dB gain (gain of 1). Over the time of the defined release time the audio signal is not manipulated anymore.

F. Pitch-Shifter

From a musical point of view, pitch shifting corresponds to the shift of a melody one or more semitones up or down. Now, from a signals point of view, pitch shifting consists of scaling the fundamental frequency and its harmonics by a specific factor.

The way this is implemented in the code is by utilizing a circular buffer that introduces every new sample and sums a value (the Shift desired) to it, thus returning a scaled version of the sample.

G. Looper

To implement this feature, we used the external flash memory to store guitar input audio and our SPST foot-switch to prompt the program to cycle through three different states as shown in the diagram below.

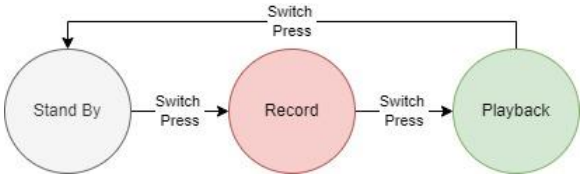


Figure 12: Looper Functionality States

During the recording state audio data gets stored in the flash memory, the next state retrieves the audio from memory to play it in a loop, and a final switch press places the pedal on stand-by state ready for the next recording. Although we were able to store and reproduce audio, it was a challenge to play it back at its original tempo due to the difficulty of synchronizing the read speed of the flash with the I2S transmit speeds. We decided to keep this feature in our design since the sped-up playback results in a very interesting effect.

VI. Board Design

We designed our PCB via the use of the AutoCAD Eagle schematic and PCB layout software. Our final PCB has dimensions of 116.51mm x 88.88mm. For simplicity and cost-effectiveness, we decided on a 2-layer design. This makes for a thin, yet robust PCB thickness. We utilized the default trace width and spacing specifications present within Eagle. When routing, we used the tRestrict, bRestrict, and vRestrict layers to prevent traces from being laid underneath our crystal oscillators (causing

interference), as well as prevent vias from being laid too close to any power pins.

For our PCB manufacturing, we decided to go with OSH Park. This company offers affordable PCBs and a simple-to-use website interface that does not require Gerber files from the PCB design software. We simply uploaded our completed .brd file from Eagle and were able to quickly and efficiently order our PCBs without complication.

VII. Acknowledgement

We would like to thank a few important people for being so integral to the success of our project. A huge thanks to: Dr. Zak Abichar, Dr. Chung Yong Chan, Don Harper, Erick Cinco, Brian Boga, Eric Brombaugh, Jim Michaud, and Darrell Murphy.

VIII. Conclusion

Every analog component was researched and tested in a simulation environment before they were used. We ordered a prototype PCB to be tested during development. After simulations were completed, we breadboarded all analog circuits with a STM32 NUCLEO Development Board for the MCU and a PCB breakout board for the CODEC. We compared changes made to the breadboard prototype to the PCB prototype. Once complete, a final PCB was designed and ordered to reflect the changes we made in the development of the prototype.

Senior Design has been an extremely valuable experience where we had the chance to apply concepts learned during our undergraduate courses. We now understand that there will always be flaws in the original design and unexpected roadblocks that manifest during implementation. As a result, we learned to overcome hurdles as a team and cooperate with one another to solve complex problems.

IX. Team Members

Diego Conterno - CpE
Interest in music technology. After graduation he wants a job in the IoT industry. Will continue to pursue his passion in music technology.

Tyler Michaud - EE
Interested in thermal optics. Currently an engineering technician at Leonardo DRS and is pursuing a career in board design.

Alejandro Porcar - CpE
Interested in the fields of aerospace and software development. After graduation he plans to pursue a career as a software engineer.

Dylan Walter - EE
Interested in audio engineering and music technology. After graduation he plans to pursue a career in audio-based hardware design.

X. References

[1] Powering Your Electronics Projects - Voltage Regulators and Converters,
https://dronebotworkshop.com/powering-your-projects/

[2] Getting Started with STM32F4xxxx MCU Hardware Development,
https://www.st.com/content/ccc/resource/technical/document/application_note/76/f9/c8/10/8a/33/4b/f0/DM00115714.pdf/files/DM00115714.pdf/jcr:content/translations/en.DM00115714.pdf

[3] Interfacing an STM32L1xx Microcontroller with an External I2S Audio Codec to Play Audio Files,
https://www.st.com/resource/en/application_note/dm00087544-interfacing-an-stm32l1xx-microcontroller-with-an-external-i2s-audio-codec-to-play-audio-files-stmicroelectronics.pdf