Final project report

EECS 452
Fall 2016

Title of Project: VE1939: A Real Time, Expressive Vocal Encoder for Musical Performance

Team members: Emma Azelborn, Kenny Carlsen, Alex Miller, Lena Sutter

I. Executive Summary

The goal of the VE1939 Vocoder was to create an expressive, playable vocoder with a suite of hardware controlled digital effects including distortion, chorus, delay, compression, and sweepable filters. In addition, the visual design and form factor of the vocoder was made a priority since these are key aspects to designing a stage-ready musical product.

Nearly all of the goals outlined in the initial project proposal were met. After the project proposal presentation, the first discussion of the scope with faculty and graduate student advisors, the compressor was deemed unnecessary and the effects in general were designated as stretch goals. By Milestone 2, the vocoder had been implemented in real time and non-real time versions of the distortion, chorus, and delay worked as expected. The sweepable filters were in the earliest stages of being implemented using library filter functions. The final version of the project had a vocoder which improved greatly over Milestone 2's vocoder, and delay and bit crush distortion are implemented and working as expected. Real time versions of the chorus and sweepable filters were built, but bugs prevented us from launching them on the effects box for demonstration.

The hardware used in the VE1939 includes two TI C5515 ezDSP chips, a Raspberry Pi 2 Model B, an Arduino Mega, a MIDI keyboard, a 7'' touch screen, and a microphone. The Raspberry Pi is used for synthesis, one C5515 does the vocoder computations, the second C5515 processes effects, and the Mega takes analog inputs from the knobs and faders that are mounted on the box.

The vocoder algorithm is not much more complex than a simple filter. First, the signal is windowed and we take the Fast Fourier Transform (FFT) of both the synthesized signal from the Raspberry Pi and the speech signal from the microphone input. The magnitude of each signal is multiplied and the phase component of only the synthesized signal is preserved. The inverse transform of this new magnitude and phase combination is taken, the window is applied again, and the result is a vocoded output.

The distortion, EQ, chorus, and delay effects are all time domain based signal processing. The distortion is a bit crusher, which is a highly efficient distortion method that limits the number of bits representing the signal. The delay is a circular buffer with offset read and write indexes according to the user specified delay time. It also includes a feedback line between the read and write indexes, allowing for multiple repetitions of the sound. The best non-real time prototype of the chorus implements a similar circular buffer and feedback schema, but the distance between the read and write indexes is sinusoidally varying. In order to prevent pops and clicks from the discontinuities this causes, linear interpolation is done between the sampled values. The EQ design uses a filter bank of three simple direct-form II biquad design with 5 coefficients per filter. The filters were designed to allow the coefficients to be updated in real time thus giving the user the ability to change filter gain and corner/center frequency for filter sweep effects.

The final VE1939 prototype performed within our initial goals. The latency of the full system was approximately 52 ms, making it entirely feasible to perform without a distracting delay to the musician. Several suitable synthesizer sounds were found to give a rich timbre to the vocoder. The largest problems the VE1939 is still faced with are aliasing and high frequency loss due to the 24kHz sampling rate and the missing EQ and Chorus effects. While the aliasing is minor and the real time EQ and chorus effects are close to complete, the high frequency loss makes intelligibility difficult on the vocoded output since fricatives have a lot of high frequency content. This problem is largely solved with a Wet/Dry knob that allows the performer to add in an amount of non-vocoded output, which has better intelligibility.

II. Project Description

The following sections will further discuss the VE1939 project goals, list the processing steps required, and then describe each of those steps in more detail.

Project Goals

A vocoder is a filter that applies the formants of a spoken or sung signal to a broadband carrier signal such as noise or a sawtooth wave. Depending on the carrier signal, the vocoder can achieve different effects. With white noise as a carrier, the vocoded output simply approximates the original speech signal. The vocoder was originally developed for this speech approximating application in 1939 [1], as it allowed for lower bandwidth telephone communication. However, the phenomena that the vocoder has become known for over time is the singing robotic voice that results from using a pitched carrier. The sound is often confused with autotune or talk boxes, but has advantages over each of these effects. Autotune is monophonic, meaning only one note can be played at once. The talkbox has a steep learning curve and involves having an uncomfortable plastic tube in the performer's mouth. The vocoder overcomes both of these problems, allowing the performer to simply sing into a microphone and play melodies or chords on a keyboard to create this unique and captivating effect.

The main goal of the VE1939 was to create a vocoder based performance system that is playable, customizable, and stylish. In order for the vocoder to be playable, the latency of the system had to be very low. Ideally, the latency would be less than 30 ms, but we were willing to allow a latency up to 100 ms for the proof of concept of the VE1939. Ultimately, our prototype had a latency of 52.52ms [Table 1]. The VE1939 allows the artist to connect their own microphones and synthesizers, which makes our system more customizable and easier to fit into existing performance setups. We put a priority on the visual design of the VE1939, creating a retro aesthetic by building the VE1939 inside of a customized ammo box [Figure 1.].

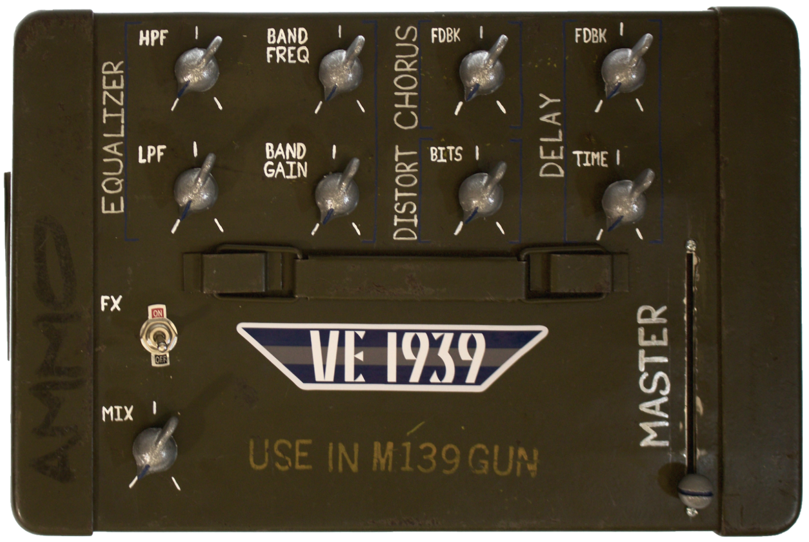

Figure 1: Finished VE1939 box using ammo box and custom knobs to create a retro aesthetic

Processing Steps

The VE1939 can be divided into a sequence of processing steps, which are shown in the high level block diagram [Figure 2]. Each step will be described in more detail in the following sections. First, a synthesized carrier signal is generated by the Raspberry Pi. Next, the carrier signal and the speech input are sent into the vocoding algorithm, which is the heart of the VE1939. The newly vocoded signal is then sent to the effects module, where it goes through EQ, chorus, distortion, and delay. The specific settings for those effects are controlled by analog knobs whose values are sent to the system with an Arduino Mega. The final vocoded and effected signal is then sent through a wet/dry mix and then to the analog output. Together, these steps create a dynamic vocoder performance system.

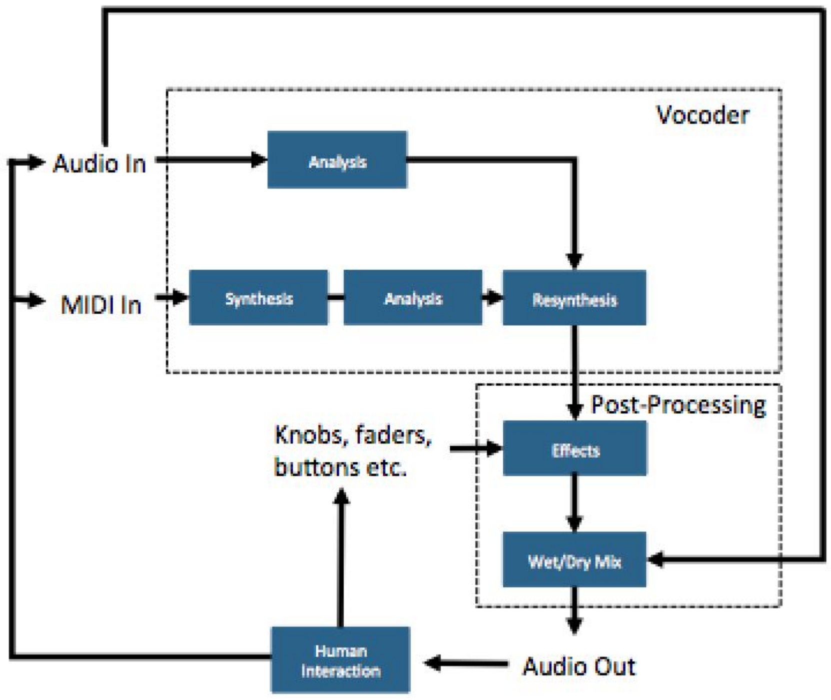

Figure 2: VE1939 High Level Block Diagram

Synthesis Engine

One of the two major components of a vocoder system is the ‘synth’ input which acts as the pitched carrier to which the formant of the vocal input is applied. While developing a synthesis engine was outside the scope of the VE1939 project this processing step is so critical to the function of a vocoder that special mention of the VE1939 synthesis engine is required. The VE1939 uses a Raspberry Pi 2 Model B as its synthesis computer, which operates outside of the main vocoder enclosure alongside the MIDI keyboard used for performance. The only major requirement for the synthesis engine was the ability to produce broadband carriers such as sawtooth wave or a square wave in order to achieve optimal vocoder performance. The engine ultimately chosen for the VE1939 was the Raspbian OS based Qsynth platform, which runs on a soundfont architecture and supports an extensive variety of sounds from the simple waveforms described above to more complicated string and choir sounds.

Vocoder

The vocoding algorithm is the heart of the VE1939. As described previously, the vocoded effect involves combining sung or spoken speech with a carrier signal, resulting in a singing robot sound. The vocoding algorithm has two inputs, a synthesized broadband carrier signal and speech. An FFT is performed on each signal to translate them into the frequency domain. Then the magnitude and phase for each signal is calculated. To apply the formants of the voice to the synth, the magnitudes of both signals are multiplied together. The phase of the synthesized carrier is used and the phase of the speech is discarded. This ensures that the phase will stay continuous, since the carrier is guaranteed to always have a continuous phase while the speech input will not. This new magnitude and phase are then sent through an inverse FFT to get the time domain vocoded signal.

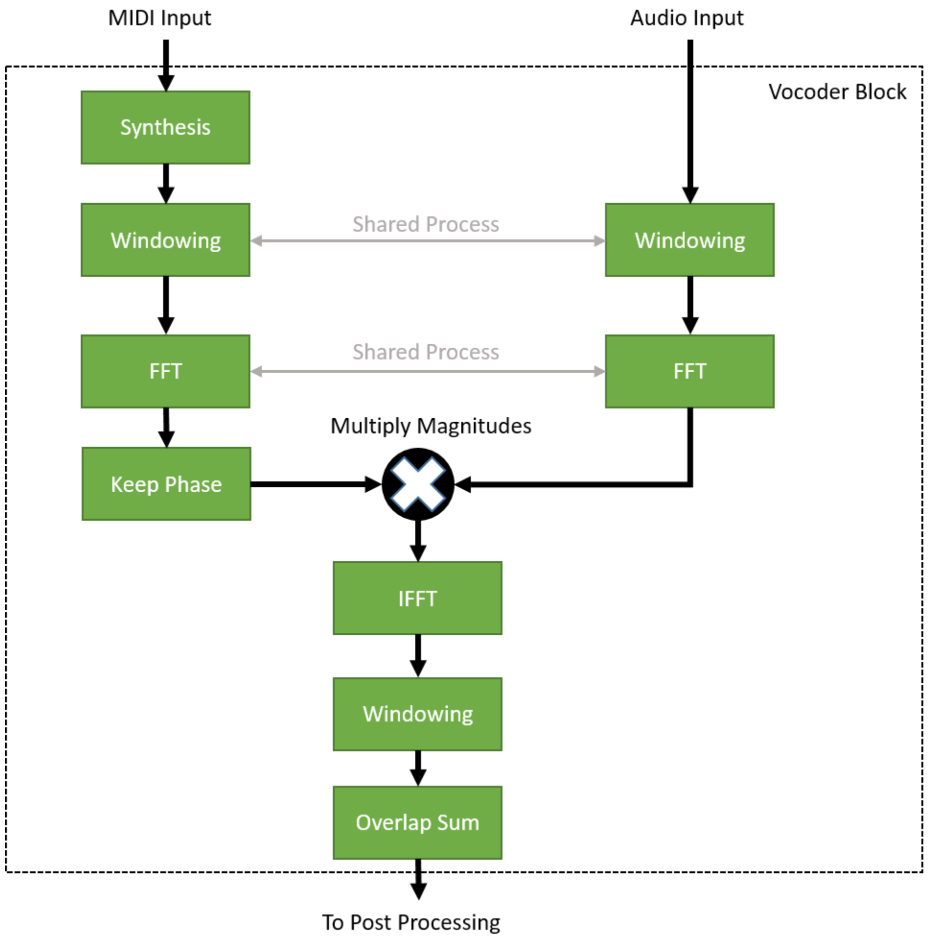

Figure 3: The Vocoder Block Diagram shows the VE1939 vocoding algorithm in detail.

Since the VE1939 is a real time performance system, the vocoding algorithm operates on one small subset of samples at a time. More specifically, the VE1939 uses an overlap add approach, where a window is applied to each group of samples processed, and each frame overlaps with the second half of the samples from the previous group. To transition smoothly between these overlapping frames, triangular windows are used. The signal is windowed before and after the algorithm with the square root of a triangle window, which ensures that, if a signal of all ones was fed in, the overall overlapping frames will always add up to one.

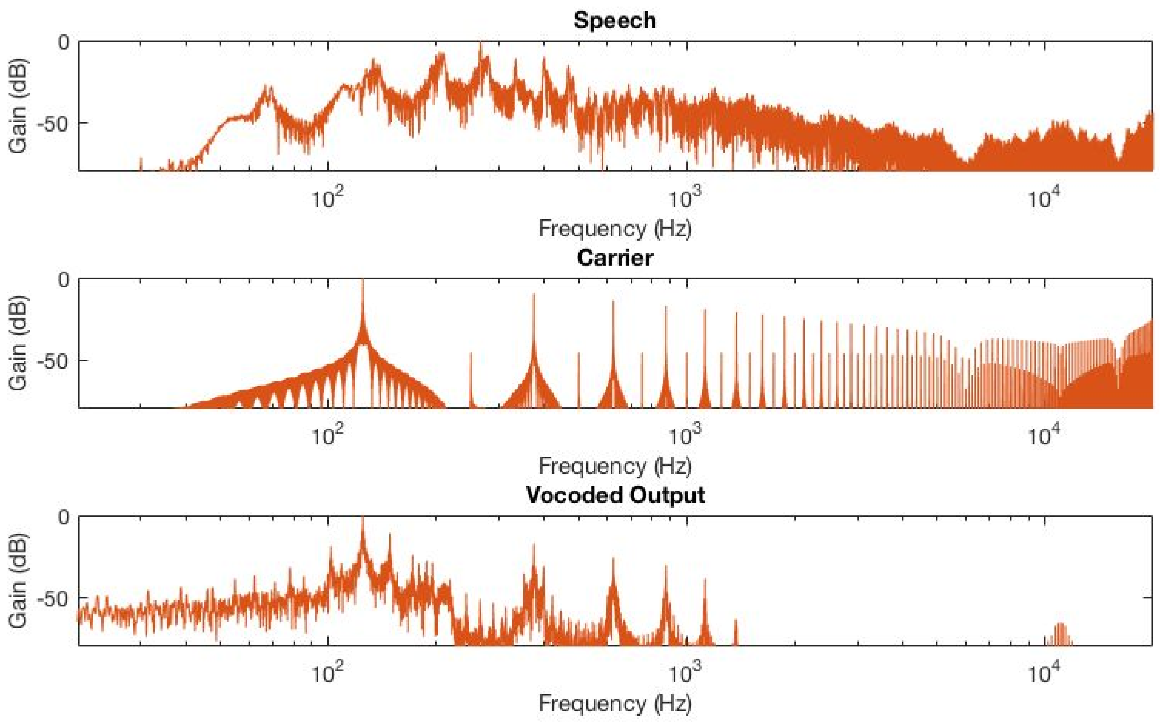

Figure 4: VE1939 vocoding algorithm applied to speech and carrier input signals

The choice of a sampling rate was a critical decision for the VE1939 as we needed to maximize our workable frequency range while still maintaining the real time invariant. Ultimately the VE1939 team decided on a sampling rate of 24 kHz, but implementing this sampling rate on the C5515 was not without complication. While the C5515 does provide the ability to alter the default input and output rates the VE1939 team was unable to successfully configure a C5515 to run at a 24 kHz input sampling rate. All attempts simply resulted in no output which is believed to be caused by a hardware configuration bug outside the scope of the project. Adapting the code to work on only every other sample acted as workaround to force a 24 kHz sampling rate, but introduced a new problem in the form of aliasing. The C5515 by default includes an anti-aliasing filter at the nyquist frequency associated with its input sampling rate, but because the VE1939 uses a sampling rate of 24 kHz without actually setting the C5515 accordingly the anti-aliasing filter is set 12kHz too high and is less effective. Unfortunately this is a trade off the VE1939 team had no choice but to take due to the apparent hardware bug.

Sweepable Filters/Equalizer

Equalization in a general is the removal of unwanted frequency content from a signal before output, and is often used in both a 'signal repair' sense (where an attempt is made to remove noise or other unwanted artifacts from the input signal) and a 'creative' sense (where the focus is more on tonally shaping the input signal). The VE1939 filter bank was designed to serve both purposes by providing fixed-frequency, variable-gain low pass and high pass filters for signal conditioning as well as a variable-frequency, variable-Q bandpass filter for creative effects.

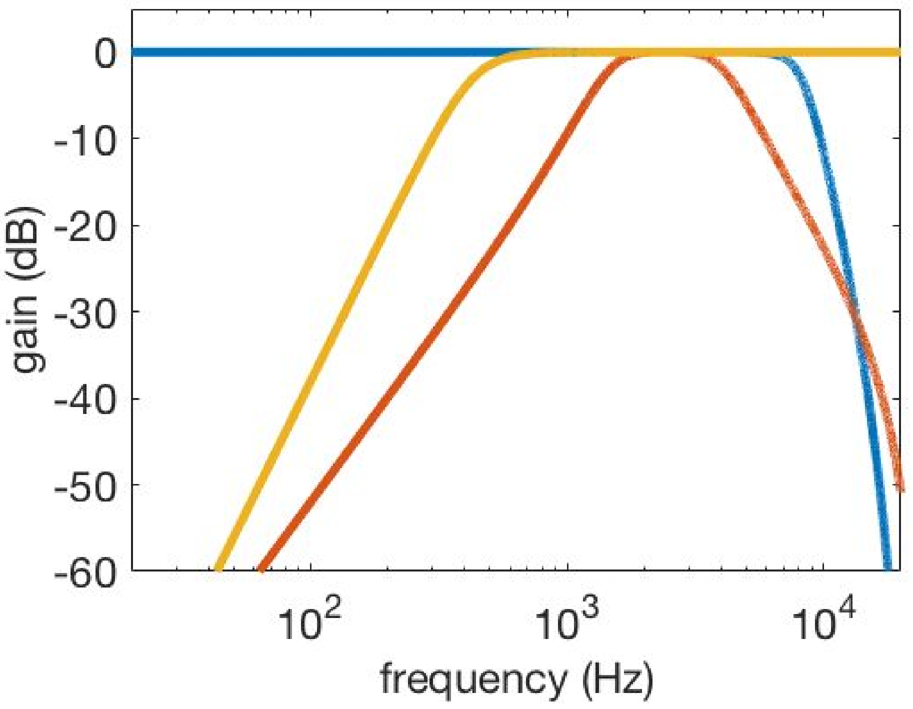

Figure 5: A prototype example of the type of serial filtering designed for the VE1939. Frequencies of interest include a HPF (Yellow) at 1.5 kHz, and LPF (Blue) at 4 kHz, and a BPF (Orange) at 2.7 kHz.

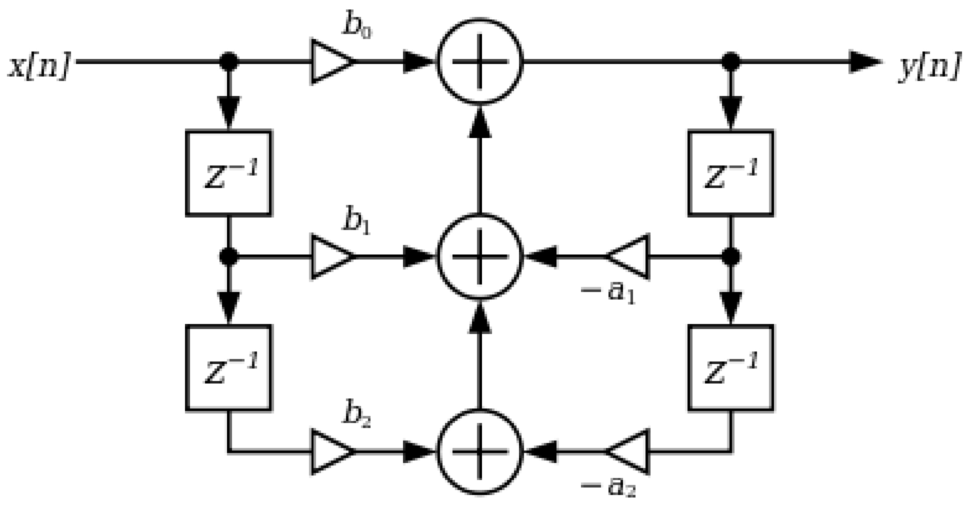

Figure 6: A Direct-Form II 2nd-order filter topology

The VE1939 equalizer bank uses three direct-form II IIR filters in series via the Texas Instruments provided iircas5 C5515 function, summarized in figure 5 as a signal flow topology. Direct-form II filters can be defined as the series of a two-pole filter section (the right side of the above figure) and a two-zero filter section (the left side of the above figure) without regard for order [2]. Each filter in the VE1939 can be separately represented by the difference equation[3]:

$$y(n) = b_0x(n) + b_1x(n-1) + b_2x(n-2) - a_1y(n-1) - a_2y(n-2)$$

where b_0, b_1 ... are filter specific coefficients calculated on-the-fly and normalized to an a_0 factor. The VE1939 was designed to allow users to directly manipulate the coefficients for each filter via knobs on the vocoder interface, but due to real time constraints and output framing issues the EQ effect was left out of the final demo vocoder. For a more in-depth description of the VE1939 EQ module and discussion of potential real time bugs see Appendix D.

Distortion

Distortion is an audio effect which alters the original shape of the waveform, usually in a nonlinear way. The distortion in the VE1939 is a bit crushing effect which achieves distortion by limiting the number of bits which represent each sample of the audio signal. When the bit depth is large, the signal is almost unchanged because only very insignificant bits are missing. When the depth is small, however, the signal is more distorted as many bits of precision have been removed. In the extreme case where the bit depth is only one bit, the signal is reduced to entirely values of -1, -0.5, 0, or 0.5. This introduces an incredible amount of distortion, which is shown in the following plot created with a MATLAB prototype. The VE1939 includes a knob which allows the user to change the bit depth in real time, ranging from 1-15 bits. See Appendix A for a more detailed discussion.

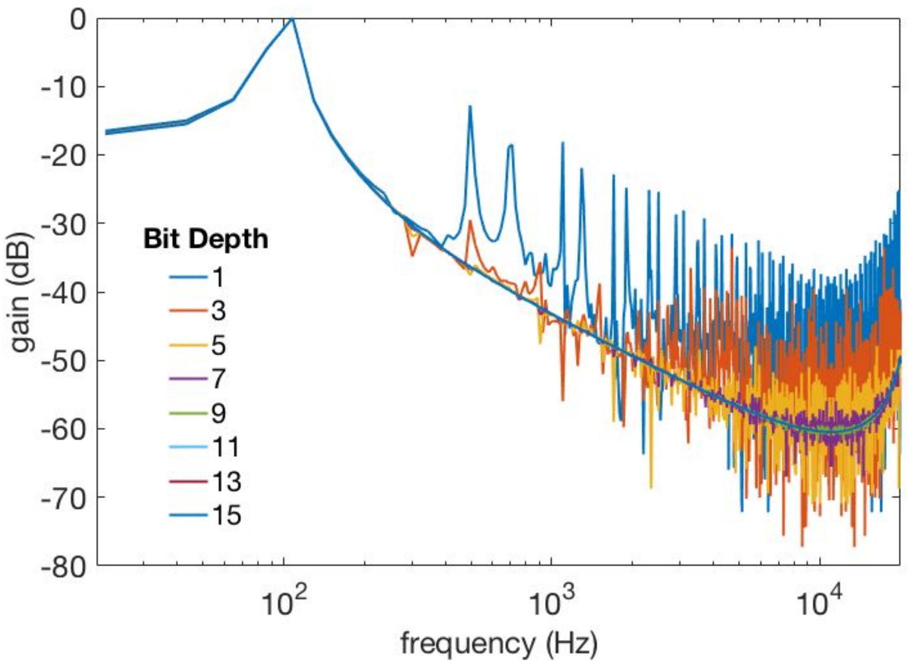

Figure 8: This plot shows the theoretical frequency response of the bit crush algorithm for a 200 Hz sine tone input at varying bit depths. It was generated in MATLAB when prototyping the bit crush algorithm.

Chorus

Chorus is a versatile effect and its sound can vary wildly between implementations. A simple chorus makes a signal thicker and richer. More complex chorus effects give the illusion of one voice becoming multiple. Chorus is defined as being a variable time delay with delay times ranging between about 10 and 30 ms [6]. This is varied either sinusoidally or with low passed noise. Multiple delay lines can be used, and chorus may or may not include feedback. For most digital choruses, interpolation is used between samples to prevent pops and clicks from discontinuities in the signal. The VE1939 chorus has a single delay line with sinusoidally varied delay and includes a feedback line (Figure 6). It uses a simple linear interpolation between samples to cope with the constantly varying delay.

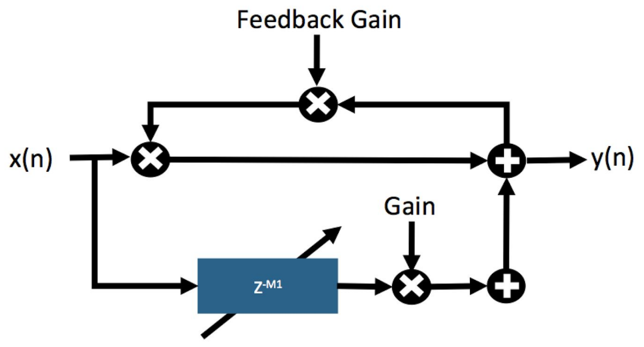

Figure 7: Block Diagram of VE1939 Chorus

Delay

Delay is an audio effect whereby the input audio is delayed by a set amount before being output. Generally, the wet signal is mixed with the dry signal to create an echo effect, and then the wet signal is scaled and fed back into the system to make the echo continuous. The way it was implemented in the system was by using a circular buffer of fixed length, and two indices that cycle through it. The first index was a write index, which took individual samples from the input buffer, added scaled feedback samples, and put the result into the circular buffer. The second index was a read index, which took samples out of the circular buffer and, added with dry passthrough samples, placed them in an output buffer. The read index was also the source of the feedback samples that were put back into the circular buffer.

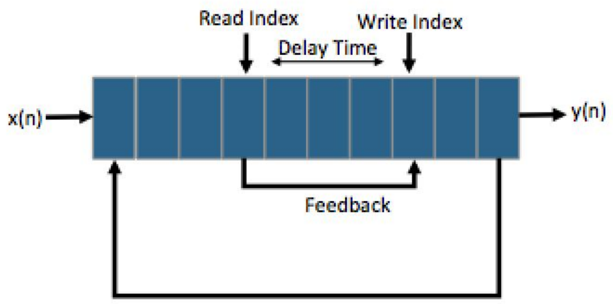

Figure 9: Delay Block Diagram

The two user controllable parameters of the VE1939 delay were feedback and delay time. The feedback value was simply converted to a Q15 DATA number before being passed into the function and used at the writing stage of the effect. The delay time was passed into the effect as an integer number of samples, and checked against the previous delay time. If it was different, then the function recalculated where the read index should be relative to the write index.

Arduino Mega and Physical Interface

A major aspect of making the VE1939 both expressive and playable was providing the ability for users to change vocoder and effect parameters 'on-the-fly'. Using an Arduino Mega as a hardware microcontroller the VE1939 is able to poll the state of hardware controls on the vocoder enclosure and transmit their value over UART at a rate of 9600 baud (or 9.6 kHz). Polling is performed applying a bias to the input device (ex: a potentiometer) and recording the voltage drop across the first stage of the voltage divider created by the potentiometer. In order to keep the system as robust as possible the Mega constantly polls for individual control states and immediately reports them without the use of any sort of C5515 to Arduino interrupt. This implementation allows the effects-managing C5515 to simply record and use the newest set of control values when necessary without requiring complicated multichip timing routines.

System Architecture Overview

The VE1939 hardware architecture consists of four SoC components and three input components namely a Raspberry Pi, a Vocoder C5515, an Effects C5515, an Arduino Mega, a MIDI keyboard, a microphone, and custom effect control knobs. The system architecture can be best understood by expanding on each component's role, proceeding from input to output (or from left to right in figure 9).

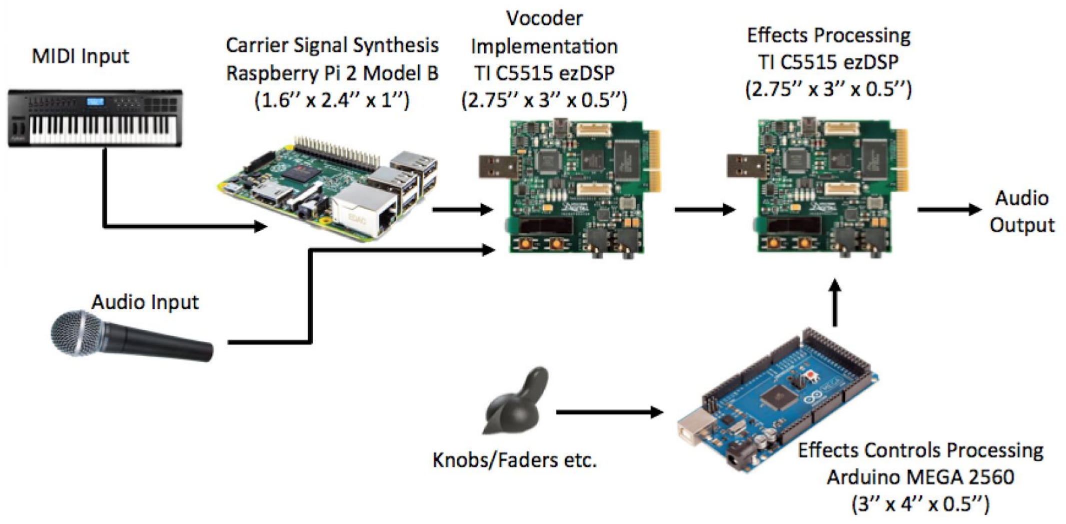

Figure 10: The VE1939 hardware system architecture diagram

<u>MIDI Keyboard</u> - The MIDI keyboard acts as a playing surface for a musician performing with the VE1939. It is connected to Raspberry Pi via a USB cable.

<u>Raspberry Pi</u> - The Raspberry Pi is the host for the VE1939 synthesis engine and performs all functions necessary for carrier signal generation. It receives input from the MIDI keyboard via USB and outputs carrier signal audio via a ⅛" audio cable.

<u>Audio Input</u> - "Audio Input" represents the vocal input signal path but can take a variety of forms depending on a particular performers setup. For the Design Expo and in-class demo the VE1939 developers used a microphone to microphone pre-amplifier to dynamic range compressor signal chain. Regardless of any situation specific setup the output of this system element must be an XLR audio cable.

Vocoder C5515 - The Vocoder C5515 is the DSP chip which provides the core functionality of the VE1939 performance system. All vocoder implementation related processing is performed on this chip. Expected inputs include audio cables from the Audio Input and Raspberry Pi stages combined into one ⅛" audio cable input via a signal combiner cable. The expected output is vocoded audio via a ⅛" audio cable.

Effects C5515 - The Effects C5515 provides the DSP for all auxiliary VE1939 functionality i.e. effects processing and related tasks. Expected inputs include vocoded audio via the ⅛" input jack and the physical interface state values via the UART Rx port. Expected output is the final system output via a ⅛" audio cable.

Knobs/Faders - Custom printed knobs installed onto potentiometers and a master fader provide performers the ability to alter audio effects in real time. Expected outputs are voltage values via wires to be read by the Arduino Mega.

Arduino Mega - The Arduino Mega manages the polling and reporting of knob/fader states to the effects C5515. Expected inputs are analog voltages from individual enclosure controls. Expected output is input control values via UART.

Parts List

Note: The VE1939 Parts List [Table 2] includes only those parts that were used in the building of the VE1939 enclosure and major DSP components. The scope of the parts list was limited in this way due to the fact that the audio and MIDI input signal chain could potentially include a large variety of different components depending on the use case. These potential variances to the input setup need not be considered in detail because the VE1939 will function as designed given input is provided correctly as detailed above.

III. Milestones

Throughout the semester, we had two milestone updates where we presented our progress. These milestone goals and presentations were a tool to gauge our progress and get feedback on our progress from course staff.

Milestone 1

Our original milestone 1 goals were to have MATLAB prototypes of our vocoder algorithm and separate effects algorithms. We also planned to have MATLAB prototypes of our individually developed carrier (synthesizer) signals.

The carrier signals were delayed and then scrapped entirely, replaced by pre-existing and freely available Raspberry Pi synthesizer software. It turned out that the carrier signals were less important than first thought, and using pre-existing software allowed us to focus on the central concepts of our project instead of MIDI integration and synthesis.

Additional Milestone 1 goals were ideally C implementations for our effects, some work on the actual C5515 chip, and work on the physical unit case begun. These goals were completed or unchanged and did not factor into the final completed work.

Milestone 2
Our original milestone 2 goals were to create a fully fledged vocal performance system with vocoder and user-controllable effects. Work between milestone 1 and 2 was to implement synthesis, vocoding, and effects on C5515s, and to build the control box.

After milestone 1 we decided to abandon synthesis and MIDI input on a C5515, and to use a Raspberry Pi synthesizer. Additionally, while we worked on the effects until Design Expo morning, we were unable to functionally get our EQ and chorus working. These were therefore left out of our demo.

The box was unfinished before milestone 2, but by Design Expo was assembled and working. The vocoding on a C5515 was completed by milestone 2, and only further tuned before Design Expo.

IV. Project Demonstration

Our demonstration at the Design Expo and to the class consisted of a self-contained vocal performance station. We had a keyboard and synthesizer to create a carrier signal as well as a microphone with a preamp and a compressor to get a good modulator signal. The VE1939 box was next to the keyboard and the microphone was handheld. This allowed a person to play the vocoder solo, or two or more people to play the vocoder and control the effects together. The audio output from the box was put through an interface and into two speakers, allowing participants and audience members to instantly hear the sound that was being processed.

Figure 11: The VE1939 team with our demonstration set up

A popular demo, and what we did in class, was to perform a rendition of Bon Iver's "715 - CRΣΣKS", wherein Lena sang while Emma controlled the synthesizer and Kenny mixed in several different effects.

V. Contributions of each member of team

Team member	Contribution	Effort
Emma Azelborn:	Vocoding Algorithm, Bit Crush Effect, Painted Box	25%
Kenny Carlsen:	Vocoding Algorithm, EQ Effect, Machined Box	25%
Alex Miller	Vocoding Algorithm, Delay Effect, Printed Knobs	25%
Lena Sutter	Vocoding Algorithm, Chorus Effect, Soldering	25%

VI. References and citations

[1] Dudley, Homer W., inventor; Bell Telephone Labor Inc, assignee. Signal Transmission. US Patent 2,151,091. March 21, 1939.

[2] Direct-Form II - https://ccrma.stanford.edu/~jos/fp/Direct_Form_II.html

[3] Pirkle, William C. Designing Audio Effect Plug-ins in C with Digital Audio Signal Processing Theory. Burlington, MA: Focal, 2013.

[4] Bristow-Johnson, Robert. "Cookbook Formulae for Audio EQ Biquad Filter Coefficients." Musicdsp.org. Web. 20 Dec. 2016. <http://www.musicdsp.org/files/Audio-EQ-Cookbook.txt>.

[5] dude837, "Delicious max/MSP Tutorial 4: Vocoder," in YouTube, Youtube, 2010. [Online]. Available: https://youtu.be/4feOFLX6238. Accessed: Dec. 2, 2016

[6] U. Zölzer, DAFX: Digital Audio Effects, U. Zölzer, Ed,. 2nd ed. Oxford, England: Wiley, John & Sons, 2011.

[7] Regan, Rick. "Decimal/Two's Complement Converter." Exploring Binary RSS. Web. 20 Dec. 2016. <http://www.exploringbinary.com/twos-complement-converter>.

[8] "MSP Tutorial 25: Using the FFT." Cycling '74. Web. 20 Dec. 2016. <https://docs.cycling74.com/max5/tutorials/msp-tut/mspchapter25.html>.

Tables

Table 1—Measured System Latency

Trial	Latency (ms)
1	50.0
2	54.4
3	48.6
4	52.2
5	57.4
Average	**52.52**

Table 2—Full Parts List

Quantity	Price	Name of Part	URL
Enclosure Controls			
10	0.95	10k Ohm Linear Rotary Potentiometer	https://www.sparkfun.com/products/9939
4	0.95	10k Ohm Logarithmic Rotary Potentiometer	https://www.sparkfun.com/products/9940
1	4.67	10k Ohm Audio (Log) 100mm slide potentiometer	http://www.mouser.com/ProductDetail/Bourns/PTB0143-2010BPA103/?qs=sGAEpiMZZMtC25l1F4XBUzCTSW%2fg3nBSFmF171TWe78%3d
3	1.99	Toggle Switch	https://www.sparkfun.com/products/9276
Enclosure Inputs/Outputs			
1	2.49	Neutrik NCJ4Fl-S Combo MONO 3-Pin XLRF/ 1/4 Inch Jack Chassis Mount with Solder Cups	http://www.markertek.com/product/ncj5fi-s/neutrik-ncj5fi-s-combo-mono-3-pin-xlrf-1-4-inch-jack-chassis-mount-with-solder-cups
2	0.99	SPF-CM 1/4-Inch Phone Female Chassis Mount Connector	http://www.markertek.com/product/spf-cm/spf-cm-1-4-inch-phone-female-chassis-mount-connector

Table 2—Full Parts List *(continued)*

Quantity	**Price**	**Name of Part**	**URL**
SoC + Peripherals			
1	45.95	Arduino Mega 2560 Rev3	https://store-usa.arduino.cc/products/arduino-mega-2560-rev3?utm_source=redirects&utm_medium=store.arduino.cc&utm_campaign=303_Redirects
2	LAB	C5515 eZDSP USB Stick	N/A
1	OWN	Raspberry Pi 2 Model B	N/A
1	79.95	Pi Foundation Display - 7" Touchscreen Display for Raspberry Pi	https://www.adafruit.com/products/2718
1	4.95	Adjustable Bent-Wire Stand	https://www.adafruit.com/products/1679
1	16.99	AmazonBasics 4 Port USB 3.0 Hub with 5v/2.5A power adapter	https://www.amazon.com/dp/B00DQFGH80/ref=psdc_281413_t3_B00TPMEOYM
Cables			
2	LAB	⅛” Audio Cable (for C5515 to C5515 and RPi to C5515 connections)	N/A
Various	N/A	¼” or XLR Cables to connect main audio input to enclosure	N/A

Appendices

Appendix A - Distortion

When developing the distortion algorithm for the VE1939, a fuzz distortion based on an exponential function was first researched. After working with the C5515 and gaining a better understanding of its capabilities, it was quickly apparent that all of our effects would be too taxing for the SoC to process in real time. Thus, the team investigated ways to make our effects as efficient as possible. Since the fuzz distortion was based on an exponential function, which itself was very slow to compute, it was not possible to optimize the distortion enough as it was. We eventually decided to change our approach and implement a bit crush algorithm instead. This choice allowed for a much faster distortion, since the bit crush algorithm is incredibly efficient. It also fits the VE1939 better than the fuzz distortion, since it is a very digital, lofi effect just as vocoding is. The choice to build a bit crush distortion instead of a fuzz distortion led to a more cohesive product which was much more efficient.

The bit crush algorithm works by limiting the number of bits which can represent the input signal. The bit depth parameter represents this number, and is specifically the number of bits past the signed bit. Therefore for a 16 bit two's complement number, the maximum bit depth is 15 bits. For any setting, $2^{(bitDepth + 1)}$ unique sample values can be represented. When the bit depth parameter is at the maximum of 15 bits, every bit is passed through and the signal is unchanged. On the other extreme, a bit depth of 1 bit means that only a single bit can represent the signal, with all 14 others always being set to zero. This limits the signal to 4 unique values: -1, -0.5, 0, and 0.5.

One of the most challenging parts of building the bit crush effect was actually implementing the prototype in MATLAB. While bit manipulations are very straight-forward in C, they are much tougher in MATLAB, especially as the given `bin2dec()` function does not support two's complement. The final MATLAB implementation involved manually changing the sign of numbers represented in Q15 format one bit at a time[7]. Below are shown two plots[Figure A. 1 and Figure A. 2] comparing the frequency spectrum of the MATLAB prototype and measured values from the final VE1939 implementation. These plots show the distortion in MATLAB and on the VE1939, featuring peaks at the fundamental frequency of the input (200Hz) and similar distortion spectrums.

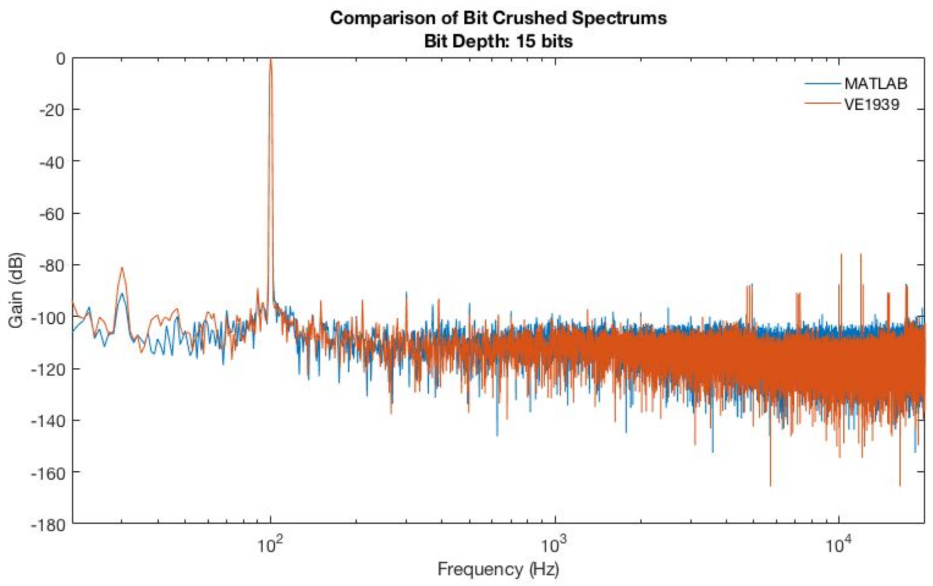

Figure A.1: Comparison of bit crushed 200 Hz sinusoid in MATLAB prototype and final VE1939 implementation for bit depth = 15

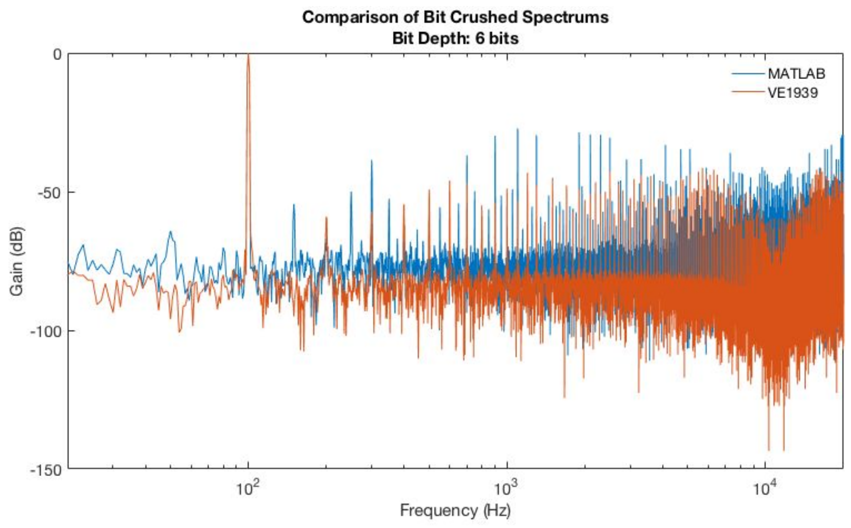

Figure A.2: Comparison of bit crushed 200 Hz sinusoid in MATLAB prototype and final VE1939 implementation for bit depth = 6

Appendix B - Chorus

Prototyping the chorus involved many decisions given the flexibility and range of applications of the effect. Therefore there were several iterations of the MATLAB version. The first utilized lowpass noise to vary the time delay and had no feedback. It also did not yet feature interpolation. The result sounded like a chorus effect, but it was fairly subtle and had pops and clicks due to discontinuities. The second iteration swapped out the low passed noise for a sinusoidally varying delay. This gave a more distinctly vintage sound that complemented the aesthetic of the VE1939 well. Feedback was then added to increase the dramaticness of the effect, since that was more computationally efficient than adding multiple delay lines in. In addition, feedback is easy to control from a linear potentiometer and varying feedback can add a wonderful dynamic aspect to chorus. Ultimately, simple linear interpolation was introduced between consecutive samples. This smoothed out the sound considerably. Once this framework was laid out parameters such as the frequency of the time delay variation and the wet/dry mix were tweaked to get the desired effect. Given that chorus is such a varied effect, there is no suitable metric for measuring the performance of our algorithm and the parameters were chosen solely based on qualitative analysis from several people trained in critical listening. The end result of these design choices is a pleasant, almost a spring reverb type chorus.

Unfortunately, implementing this version in C proved to be a challenge. The MATLAB version that we favored took too much time to run without using vectorized functions and, given that TI’s DSP library had proven unreliable, we were unable to optimize it to a point where the chorus would be able to run alongside other effects. The feedback was omitted from the algorithm and we were able to run the simplified version on the C5515. However, this version still dropped frames with enough regularity that the chorus effect was overpowered by loud clicks and it was not suitable for demonstration.

Appendix C - Delay

The MATLAB prototyping stage of the delay was quite different from the final algorithm that made it onto the chip. A variable buffer size was used with a single index instead of two indices in a circular buffer. However, this was unreasonable on the C5515 for several reasons. The delay time would not have been able to be smoothly adjusted live, and was more inefficient as well.

Despite using two indices, the way they were cycled through the circular buffer was efficiently done through the use of always incrementing and anding their values instead of using if statements. However, this limited the size of the circular delay to powers of two, and given the C5515s limited memory may have been an issue. Fortunately, a buffer size of 32768 (2^15) DATA values was easily obtainable, which gave a delay time of slightly more than 1.5 seconds. In testing, buffer sizes of up to 50000 were able to be built and run on a C5515, but 65536 (2^16) was too large. In a way, the sampling rate of 24 kHz worked to the delay's advantage, in that the same size of a buffer was able to hold a longer delay.

The sound of the delay was quite clear with no fundamental, audible issues. The biggest issue was the same as the rest of the vocoder, being the scaling of the signal and sensitivity to high and low gain. Since the VE1939 was already having issues with low gain and noise, there was no scaling done to the dry signal, and the delay was simply added on. With short delay times and high feedback this could cause serious clipping and unwanted distortion, but only at extreme settings. As such, feedback was limited to a maximum of 90%, and any remaining distortion was determined to be less harmful than losing volume through scaling.

Analyzing recorded audio samples with a controlled input, we can see that the delay time for a 24000 sample delay is correctly 1 second. Furthermore, the feedback value was set to 50%, and the first delay is correctly half the size of the original signal (0.062 V to 0.031 V).

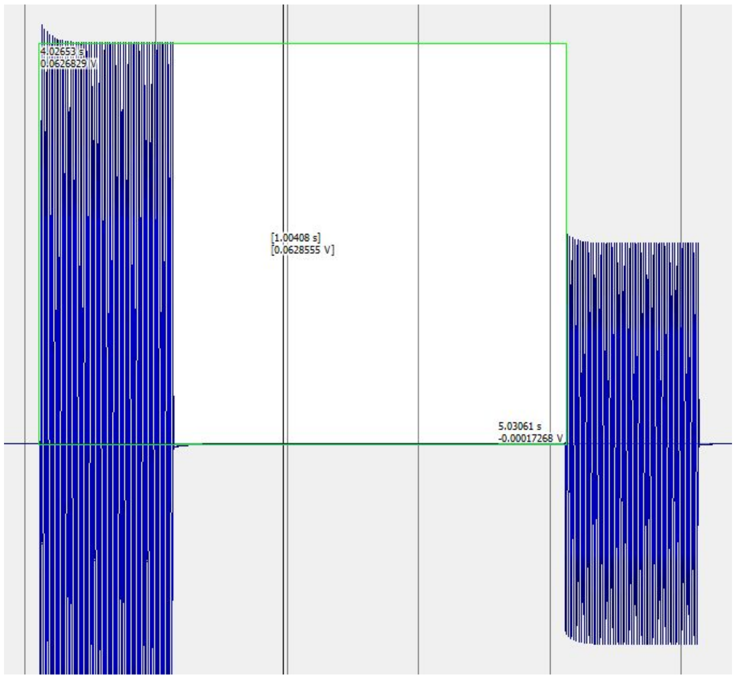

Figure C.1: Example of delay set to 1 second and 50% feedback

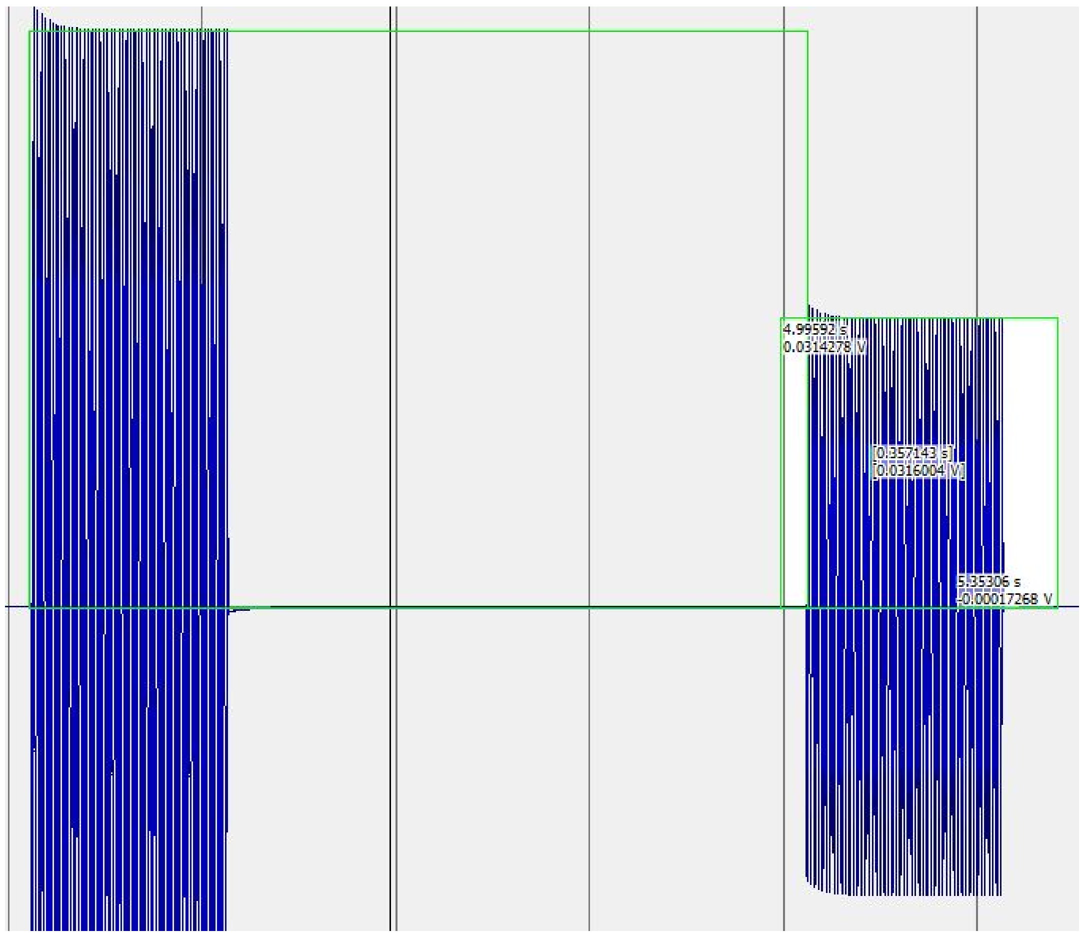

Figure C.2: Amplitude of delayed signal is 50% of original

During testing, there were several distinct effects that a simple delay could enact when a vocal sample was put through it. Below 240 samples of delay (10ms), a comb filter effect was very noticeable, with the feedback roughly controlling the "wetness" of the effect. Between 240 and 960 samples (10ms and 40ms), it still sounded filtered, but depending on the feedback (between 50% and 80%) sounded like a bad reverb. At higher (>80%) feedback values, it sounded like a flutter echo. Above 960 samples, but below 1920 samples (40ms and 80ms), the flutter was the primary effect. Above 80ms, it sounds more and more like a simple delay.

Appendix D - Sweepable Filters/Equalizer

At the outset of the VE1939 project the goal behind having an EQ module was the ability to perform filter sweeps while performing with the vocoder. A bank of three filters was deemed appropriate and in order to keep the number of front panel controls below a reasonable limit the decision was made to only allow real time center frequency variation (in addition to variable-gain) on the mid-band. High and low pass bands were slated to be delegated to fixed corner frequencies (approx. 300 Hz and 9 kHz respectively) with variable gain to allow users to control the attenuation at the edges of the frequency spectrum.

Mathematical Background

In order to allow the VE1939 EQ module to have real time adjustable filters a Direct-Form II biquad approach was taken (see “Sweepable Filters/Equalizer” for a general overview of this topology). Each filter was comprised of five coefficients which were updated in real time according to changing knob values and the filter dependant biquad coefficient equations [4] seen below. Note the inclusion of two sets of midband equations - this is a consequence of a mid-semester design change that will be discussed further below.

General Direct-Form II Biquad Transfer Function

$$H(z) = ((b_0/a_0) + (b_1/a_0) * z^{-1} + (b_2/a_0) * z^{-2}) / (1 + (a_1/a_0) * z^{-1} + (a_2/a_0) * z^{-2})$$

Intermediate Variables

$$w_0 = 2\pi * f_0/F_s$$

$$\alpha = sin(w_0) / 2 * Q$$

$$A = \sqrt{10^{G/20}} \; where \; G = gain \; (for \; peaking \; BPF \; only)$$

Table D.1: Coefficient equations for filters used in the VE1939

LPF Coefficient Equations	**HPF Coefficient Equations**	**Constant 0dB peak gain BPF**	**Peaking BPF**
$b_0 = (1 - cos(w_0) / 2$	$b_0 = (1 + cos(w_0) / 2$	$b_0 = \alpha$	$b_0 = 1 + \alpha * A$
$b_1 = 1 - cos(w_0)$	$b_1 = 1 + cos(w_0)$	$b_1 = 0$	$b_1 = -2 * cos(w_0)$
$b_2 = b_0$	$b_2 = b_0$	$b_2 = -\alpha$	$b_2 = 1 - \alpha * A$
$a_0 = 1 + \alpha$	$a_0 = 1 + \alpha$	$a_0 = 1 + \alpha$	$a_0 = 1 + \alpha/A$
$a_1 = -2 * cos(w_0)$	$a_1 = -2 * cos(w_0)$	$a_1 = -2 * cos(w_0)$	$a_1 = -2 * cos(w_0)$
$a_2 = 1 - \alpha$	$a_2 = 1 - \alpha$	$a_2 = 1 - \alpha$	$a_2 = 1 - \alpha/A$

Implementation Details and Design Challenges

All filters were implemented in real time on the C5515 using the built-in TI iircas5 function. The iircas5() function expected inputs were five filter coefficients scaled by the a_0 coefficient (as shown in the transfer function above). All calculations were performed on the C5515 as Q15 DATA integers. Coefficient overflow was a significant challenge in filter implementation but was ultimately handled by sacrificing precision for increased range by converting the offending Q15 number to a Q14 number before being passed to the iircas5 function.

As mentioned above the design of the EQ module and specifically the midband underwent significant changes towards the end of the semester. While the original plan was to use a peaking EQ to create the boosted midband normally used for filter sweep effects an alternative plan was devised due to the following:

i. Peaking EQ calculations required a log scale gain calculation which was likely would cause the EQ algorithm to be too slow for real time
ii. Due to gain scaling effects of the iircas5 function only a very small peak (~1dB) was obtainable before filter overflow distortion occurred

The solution was to use a 0 dB peak gain bandpass filter with variable Q instead of variable gain. The variable Q design works based on the idea that a large boost at a particular frequency of interest can be approximated by attenuating all frequencies *around* the frequency of interest, resulting in similar relative gain per frequency. Rather than increasing mid band filter gain users instead increase the Q which 'tightens' the filter response around the center frequency, producing a more stark response when the filter is swept across the frequency spectrum.

MATLAB Prototyping and Theoretical Response

In order to demonstrate the effectiveness of the 0 dB peak gain bandpass filter technique MATLAB filter bank prototyping was conducted. For the purposes of this prototyping the spectrum was bandlimited via the LPF and HPF to a more easily plotted 1.5 kHz to 4 kHz range. 0 dB peak gain BPF were then added at a variety of Q values and the system response was plotted, shown here in increasing Q order.

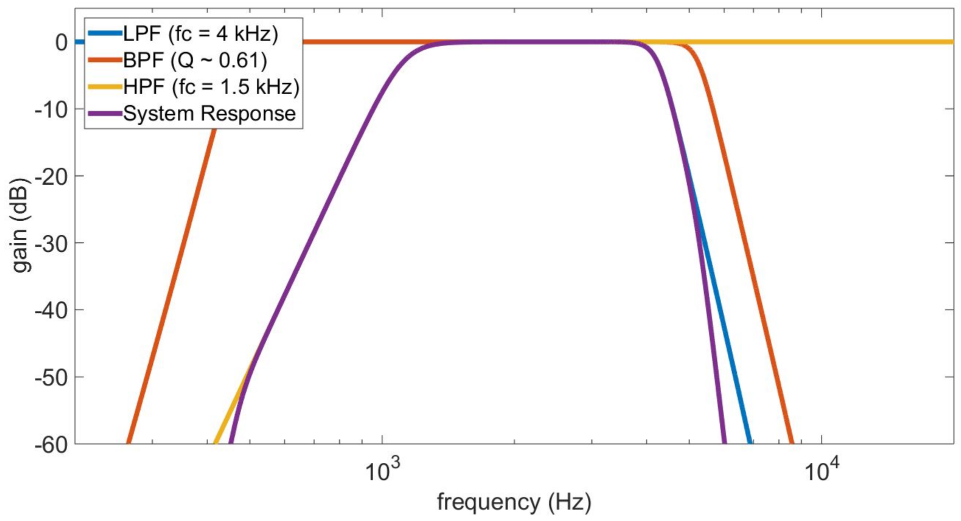

Figure D.1: With very low Q values the system exhibits a very flat response between the band limiting LPF and HPF

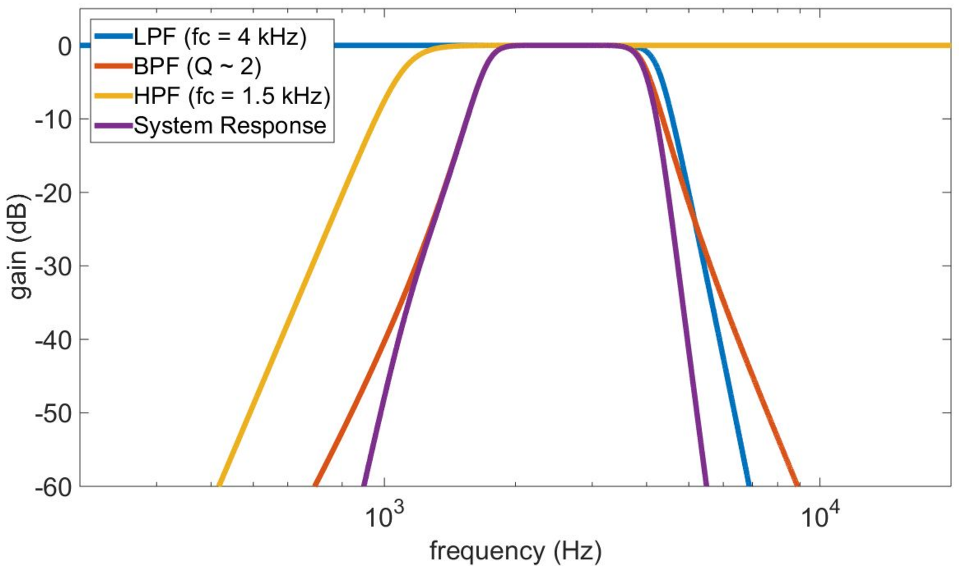

Figure D.2: Increased Q values begin to tighten the system response, with intermediate values giving a response influenced by all three filters

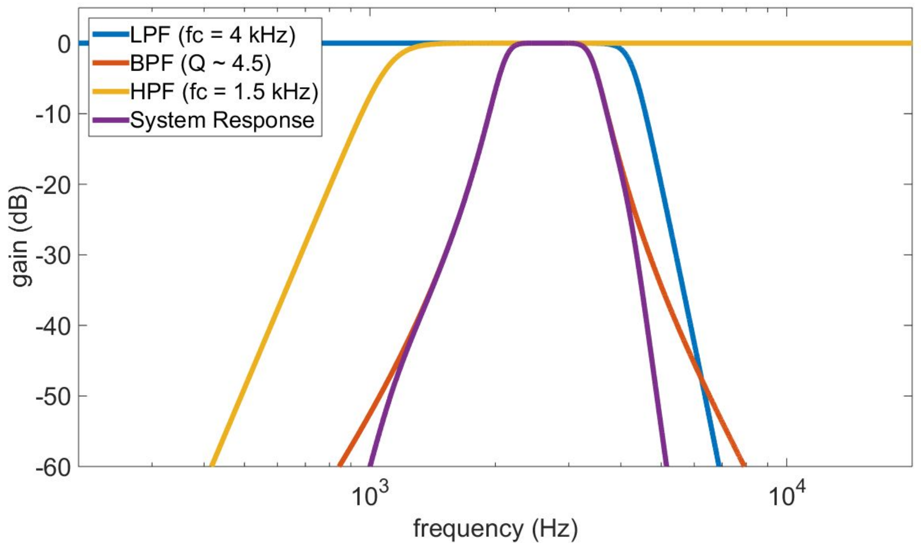

Figure D.3: With a higher but still intermediate value the influence of all filters is apparent but the BPF begins to dominate the response

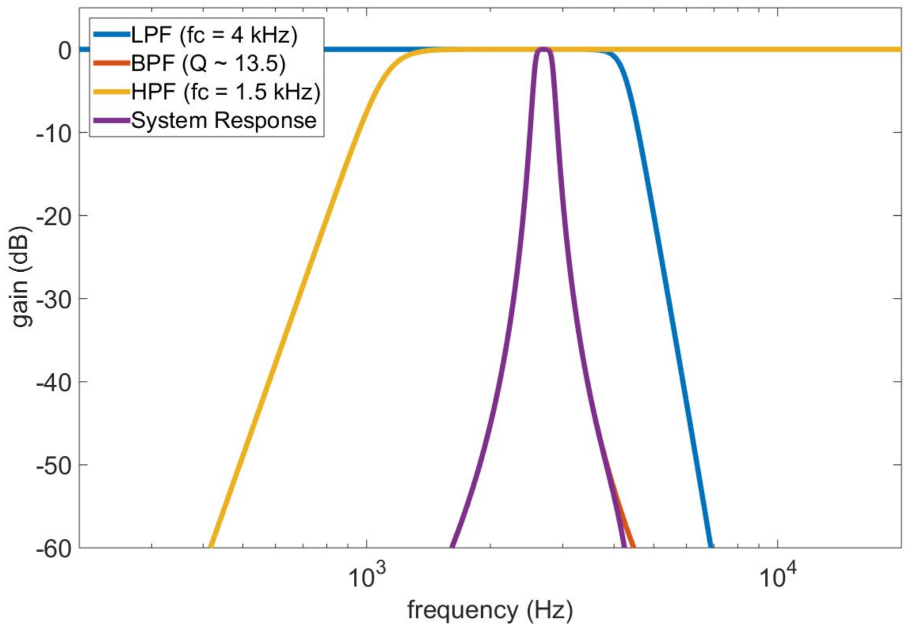

Figure D.4: At extreme Q settings the system response is completely dominated by the BPF and is extremely spectrum limited.

By allowing the user complete control over the Q factor of the mid band filter the VE1939 EQ can be used for applications ranging from subtle frequency shaping to dramatic, tight frequency sweeps across the spectrum. While perhaps not the maximally dramatic design, the 0 dB peak gain bandpass filter provides core functionality for frequency sweep effects over the range required by the VE1939 without risk of overflow.

Results
Unfortunately, the VE1939 EQ module was non-operational in our final production version of the VE1939. While the effect was successfully implemented on its own in real time as detailed above there were unreconcilable errors introduced when the effect was incorporated into our master effects suite project. Project completion timeframe constraints prohibited detailed analysis of effect failure points, but qualitatively speaking the effect was unable to produce desired filtering without the introduction of additional noise. While debugging the effect the VE1939 team discovered bizarre framing issues where pieces of previous frames would be deposited into the next frame. These observations point towards a disagreement between EQ frame processing and the frame overlapping at the project level, but further analysis has not been completed.

However, despite the unsuccessful attempt to integrate the EQ module into the master effects suite, performance of the effect in real time was still partially verified by evaluating the effect in its standalone configuration. In limited test cases, the filters performed as expected (parameters were similar to the Q ~4.5 MATLAB prototype above) and were able to operate within real time constraints.

Appendix E - Additional Vocoder Information

The vocoding algorithm is the heart of the VE1939. Before implementing it on the C5515, our team did extensive prototyping in MATLAB to develop our algorithm. The earliest prototype did not include overlapping frames. This led to lots of clicks due to discontinuities across boundaries of fft frames. To remove these discontinuities, our team started investigating an overlap-add approach which involves overlapping fft frames. With this approach we could window each frame, transitioning smoothly to the edge of the frame and removing the discontinuity on the edge. By processing overlapping frames and then adding the outputs together, there is no longer a hard switch from one frame to the next, creating seamless transitions.

One challenge of the overlap-add approach was choosing the best window for the task. There are many industry standard window shapes, each optimized for different tasks. For our application, we needed a windowing scheme that would keep the overall signal the same level. This means that if a signal of all ones is passed into the system, a signal of all ones needs to come out. Any variance means that the scaling is not consistent and would result in periodic amplitude modulation, which is undesirable. After experimenting with a number of window shapes, we ended up modeling our windowing scheme after the one used in Max/MSP's real time fft[8]. We window our signal with the square root of a triangle window before and after processing, which results in an even and consistent overall sum when subsequent frames share half of the samples.

One significant feature of our prototype vocoding algorithm which was not implemented in the VE1939 was fricative detection. Fricatives are noisy consonant sounds essential to enunciation. When using only a pitched carrier signal as has been described in this report, these consonants get muddled and lost, resulting in a slightly garbled vocoded sound. It is tough to understand the words or lyrics in the original speech. To combat this problem, our team developed a fricative detection algorithm based on zero crossings, and when fricatives were detected we switched the carrier from the pitched synth to purely white noise. This made the artist's diction much more apparent, keeping the original fricatives much closer to their original sound. Unfortunately, this feature is not implemented in the VE1939. Due to hardware limitations, the C5515 can barely keep up with the sampling rate even without this feature, and adding this feature would have required dropping the sampling rate even lower. At that point, any high frequency content that would have been added by the noise carrier would have no effect on the output because the nyquist frequency would be so low. If the VE1939 is ever produced as a consumer product, our team recommends using a more modern chip which is powerful enough to include this fricative detection feature.

Appendix F - Enclosure Build

The VE1939 was built into a vintage metal ammo box that was customly machined and painted by team members. An original logo and knobs were also added to create a one-of-a-kind look.

After being sourced, the layout of the gear in the box was drawn up to flow in an intuitive way from input, through a wet/dry mixer, then the effects, and then to the output. Marks were then made for the holes in the top and sides. In addition to being drilled, internal reinforcements in the box needed to be molded so that the knobs could reach the outside.

Once everything was installed, the labels penciled in and then carefully hand painted for the controls. Measured tick marks were also painted around the controls to allow users to have an easier time remembering their settings.

Custom knob covers were also created for the VE1939. Precisely measured to fit the ordered potentiometers, they were first modeled in SolidWorks before several prototypes were 3D printed. At first, fittings were difficult and the design deemed unwieldy. Within a few revisions acceptable prints were created.

A logo was also designed for the VE1939. Inspired by WWII era fighter planes, it has angled wings in silver and blue with the ‘VE 1939' name stamped across it. Several iterations were gone through to find the correct combination of colors and style that then informed the rest of the design decisions.

Knobs were soldered and run into a breadboard circuit that allowed us to separate the controls for each parameter while also grouping them so we could have a bypass switch that turned off power for the effects potentiometers. This set their values to 0 and let the signal go through the effects C5515 without being changed. The wet/dry potentiometer was on a different loop, however, and could be used regardless of how the bypass switch was set.

Input on the left (XLR) and output on the right (¼” TRS). Jacks and necessary cables were measured out and soldered to minimize waste of space.

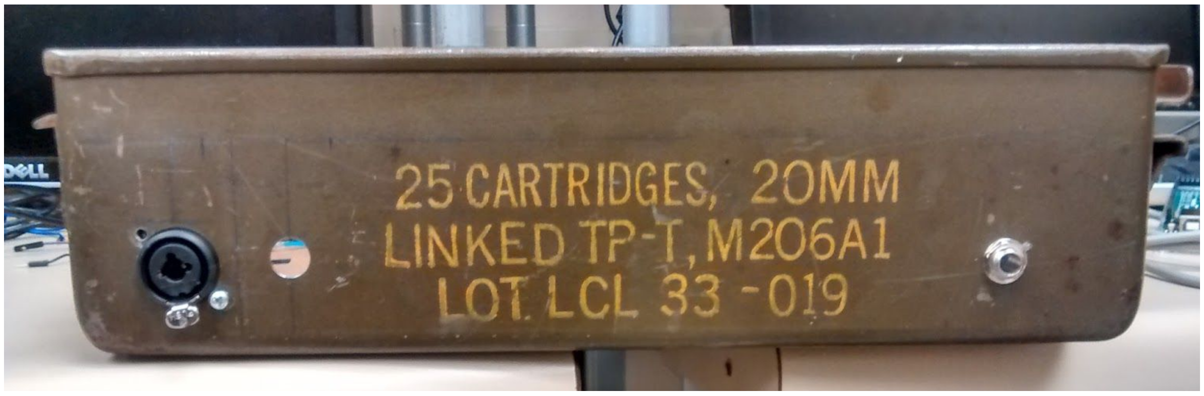

Figure F.1: Input and Output Jack on the box

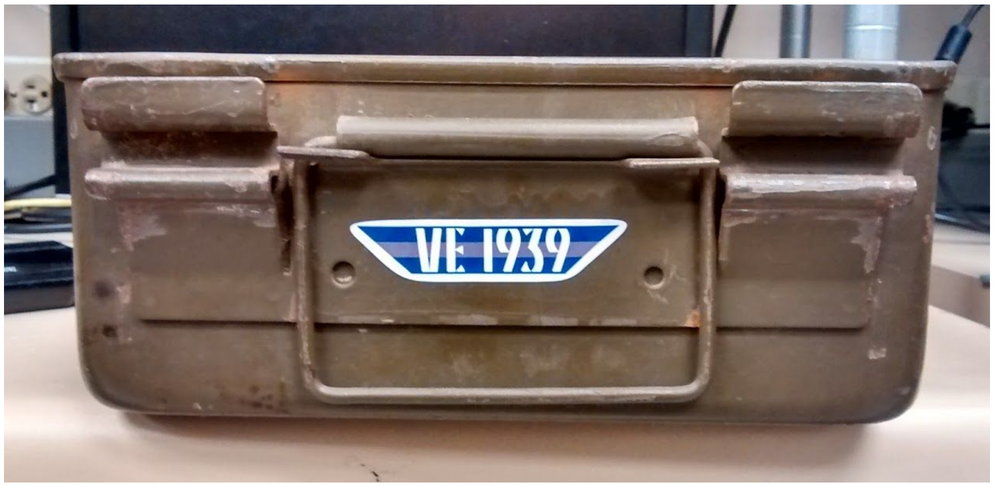

Figure F.2: Logo example

Appendix G - External Equipment

To help narrow the focus of the VE1939 project, several pieces of external hardware were used. These were not used in place of any effects or processing that we ever planned to do, but allowed us to focus on what did fall within the scope of our original plan.

Our first input was a Shure PE15L Cardioid Dynamic microphone, with a built-in switch. This allowed us to be able to reject unwanted noise beyond what the performer was directly saying, and to be able to quickly turn off the input if things went wrong.

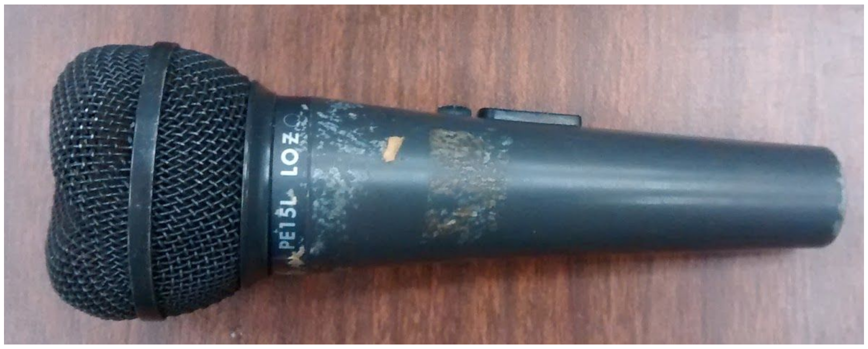

Figure G.1: Microphone with switched used for vocal input

On the input side, we used a preamp and compressor to allowed us to keep the input loud enough to make it through the vocoding processes, but compress it to keep it from clipping. The equipment used was an M-Audio Duo Mic Pre Stand Alone A/D Converter and a Symetrix 501 Compressor, pictured here.

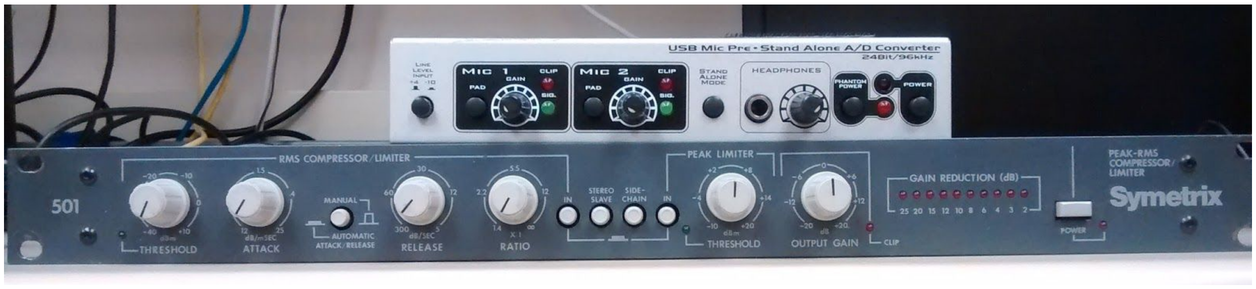

Figure G.2: Preamp and Compressor used for vocal input

For the synthesizer, we used an M-Audio Axiom49 USB MIDI keyboard plugged into a Raspberry Pi. We used such a relatively large keyboard because it more closely reflected the range of a human singer, more so than an octave and a half keyboard.

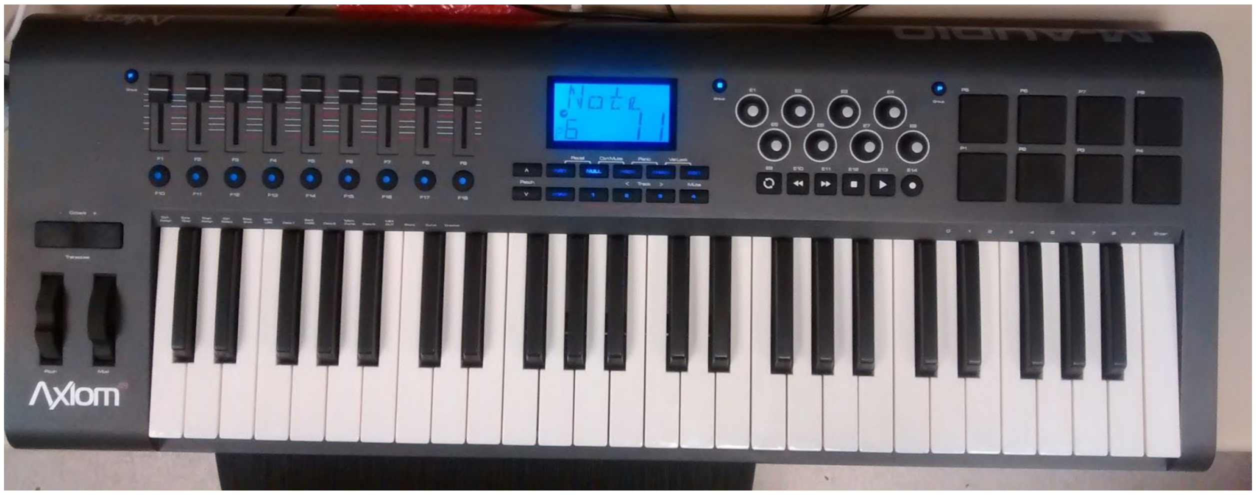

Figure G.3 MIDI Keyboard used for Synthesizer input

On the output side, we used a Rane LT22 Line Transformer to condition the signal and prevent hums and other external noise. This then fed two Equator D5 Direct Field Monitors, which provided us slightly more volume and much higher fidelity than the lab speakers.

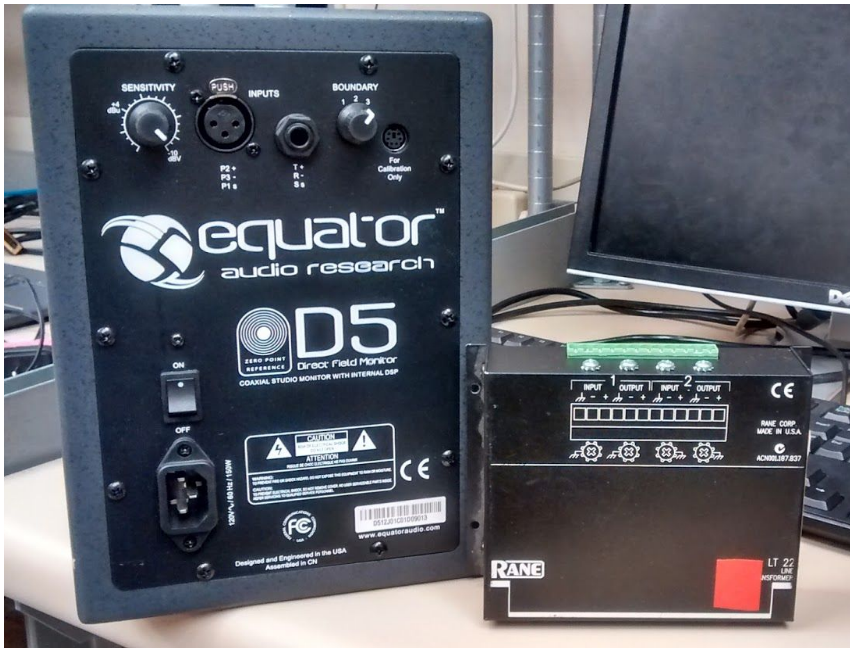

Figure G.4: Speakers and Isolation Transformer

The downside to the majority of the external equipment used was simply greatly increasing the size and complexity of our setup, going from a medium sized ammo box to a full tabletop of gear. However, the flexibility that accommodating a variety of input and output signal chains affords means the VE1939 could potentially fit into more live setups and existing musician rigs, making the benefits of the external gear far outweigh the costs.