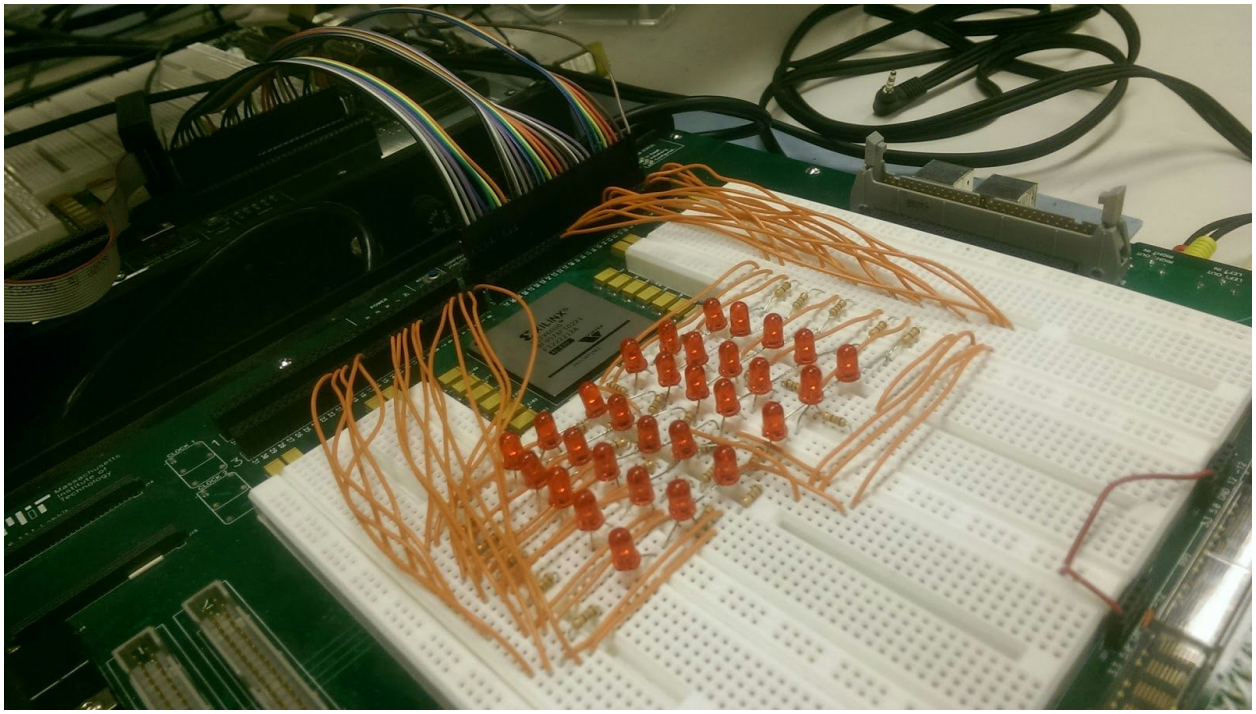
FPGA DJ

6.111 Final Project Report

Alex Sloboda and Madeleine Waller



Abstract

The FPGA DJ system is a music synthesizer capable of taking in two audio signals via a 3.5 mm jack, processing them in real time, and mixing them to produce a single handcrafted output consisting of surround sound and an LED volume display. The user has the ability to select the processing effects of the two audio signals individually, including both time and frequency effects - namely echo, reverse echo, filtering, and equalization. The user can then custom mix the processed signals to produce a unique audio output to the surround sound speakers via an AC97 audio codec, either selecting to output just one of the processed audio inputs, a weighted addition of the two processed audio inputs, or one of the processed audio inputs controlled by the beat/volume of the opposite audio input.

Our project is exciting because we wanted to learn about and implement signal processing techniques, and because all of the effects and synthesis occur in real time with minimal delay. Additionally, the system is not preprogrammed with effects - it allows the user to choose from a wide array of processing techniques, mixing and matching them in a variety of different combinations, adding a great layer of complexity and potential to the user experience and the system itself.

Table of Contents

High Level Block Diagram - Alex and Madeleine

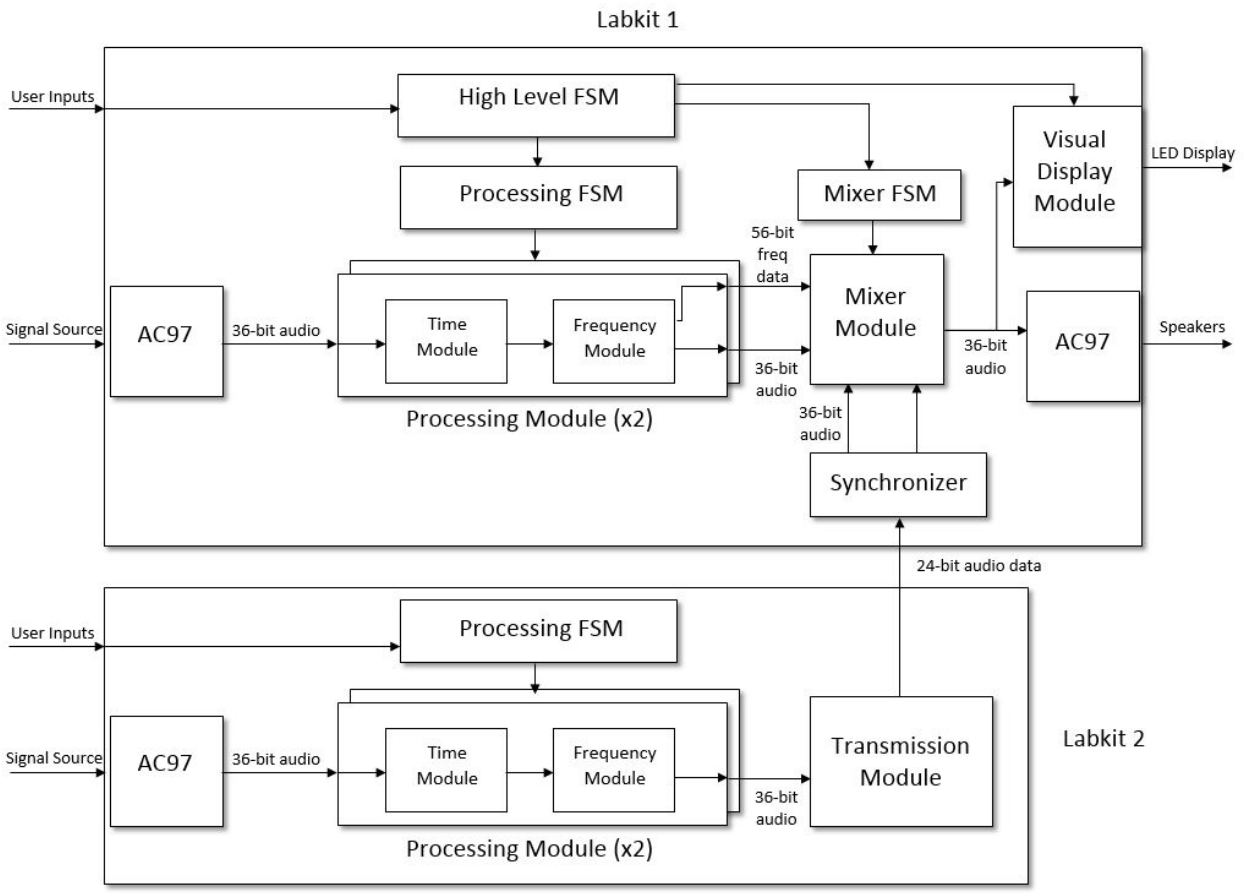


Fig. 1 System Block Diagram

Major Modules - Alex and Madeleine

AC97 Audio Codec -Alex

The AC97 audio codec serves as the input and output of our system. Via the RCA inputs on the labkit, we receive the analog audio signals from the desired audio source. The codec then performs an A-D conversion on the the analog audio signal, transforming it into a stereo output consisting of two 18 bit digital audio signals. At a speed of 48kHz these audio signals are then presented to the system along with a ready signal which signifies when the data is ready and when the AC97 is ready to receive data for output. This ready signal is used in many of our modules as a clock with which we increment audio samples through the system.

Setting up the AC97 in this manner was done by taking the modules given in Lab 5 and modifying some of the register bits sent to the codec during initialization, thereby transforming it from an 8-bit monaural source to an 18-bit stereo source. Following all of our processing, the resulting 18-bit stereo signal is presented back to the AC97 on the ready signal where it is then transformed back into an analog output and played through a pair of speakers.

Processing Module - Alex and Madeleine

Time Module - Alex

The effects carried out in the time module are time based and implemented utilizing feedback or feedforward paths around a delay block. The delay block is implemented via BRAM, performing the function of a circular buffer. As new signal data comes in it is written to an address in memory directly behind the location currently being read. Only once the read pointer passes through the entire BRAM address space will it read this new data, thereby creating approximately a 350ms delay. As built, it is capable of generating reverse echo or continuous echo, or both.

When reverse echo mode is activated, an attenuated version of the incoming signal is fed around the delay block directly to the output, creating a quieter copy of the actual signal playing ahead of the full volume version. In continuous echo mode, the output of the delay block is given an attenuated feedback path to its own input. As a result, the signal that comes out repeats a fixed amount of time later, at a quieter volume on top of the normal signal. This continues for many cycles until the attenuation fully decays the

echo. The user is able to actively adjust the attenuation factor along this continuous echo path, allowing for a wide range of echo longevity and intensity while the audio is playing.

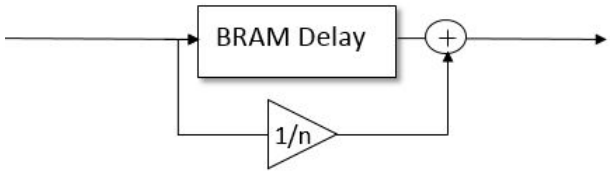


Fig. 2 Reverse Echo System Diagram

As mentioned, the continuous echo and reverse echo can be used alone or in conjunction with one another. When used together, the result is very similar to a reverberation effect as many differently delayed versions of the same sound play next to each other. This can either be overwhelming or can greatly enhance the sound of certain songs. Because of how the effects are implemented, transitions between them are seamless in the sense that there is no cut in the audio, rather the effect either comes in or fades away on top of the unprocessed audio signal.

From the perspective of the user, transitions between the effects are implemented by flipping switch 0 to on, in order activate continuous echo, and/or by flipping switch 1 to on, in order to activate reverse echo. These effects only are fully activated after the corresponding switch has been flipped and the user hits the enter button on the labkit. Whether the continuous echo is on or off, its attenuation factor can be controlled by hitting the left and right button on the labkit. This allows the user to prepare either a weak or strong attenuation factor prior to activating the effect. In any of the possible modes that the user sets up the time module, the final time processed output is then sent onto the frequency module as two 18-bit stereo signals once more.

Frequency Module - Madeleine

The frequency module takes as input the 18-bit audio data passed in from the time processing module, performs filtering and equalization on the audio signal, and outputs the 18-bit audio along with 56 bits of frequency data to the mixer for audio output to the AC97 Codec with custom volume control as described in greater detail in the mixer module.

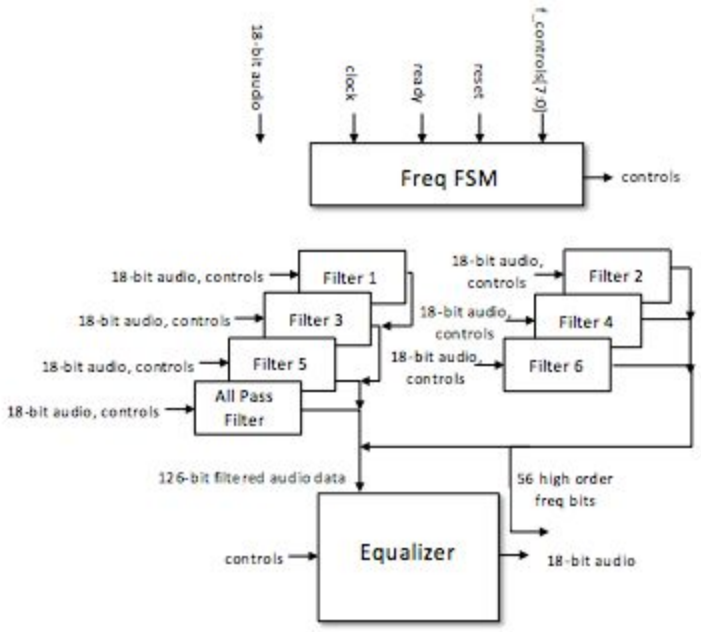


Fig. 3 Frequency Module Diagram

The frequency module consists of a high level module that instantiates six different 401-tap FIR filters and their coefficients (along with one all-pass filter to add a delay to the original signal to realign the phase for equalization), effectively dividing the audio signal into six different frequency bins. The frequency module then passes this data to an equalizer submodule that enables the user to select the effects they desire, e.g. boosting the bass in a song to make it more acoustically appealing. The labkit user interface is designed such that the user has a high degree of freedom in terms of what he/she wishes to hear. The frequency-based effects that can be selected range from outputting the original untouched audio signal to outputting the audio of just one of the frequency bands or, perhaps most importantly, outputting an equalized audio output - a weighted combination of the various frequency bands as selected by the user via buttons on the labkit. This feature provides the user with the ability to boost the frequency ranges of songs that they deem most desirable and pleasing to the ear.

As mentioned above, the frequency module instantiates six individual filters with differents sets of coefficients to perform the digital filtering for each frequency bin. The individual filter module is a 401-tap FIR filter that takes an 18 bit audio signal in the time domain as input at each ready pulse and returns an 18 bit filtered audio signal as output. The filter is designed to amplify a specific frequency range while attenuating all frequencies outside of this range. Thus, with a 48 kHz ready pulse and a 27 MHz clock, we use 401 out of the 562 available 27 Mhz clock cycles that occur between ready pulses

to perform the filtering (we perform one multiplication on each clock cycle before adding the weighted samples). I divided the audio spectrum into the following frequency ranges:

- below 120 Hz (low bass)
- 120 Hz - 260 Hz (mid/upper bass, male voice)
- 260 Hz - 600 Hz (lower midrange)
- 600 Hz - 1200 Hz (mid range)
- 1200 Hz - 4 kHz (upper mid range)
- above 4 kHz (high frequencies)

Shown below is a Matlab plot of my 401-tap FIR filter with a 120-260 Hz passband, which I generated to ensure the validity of the FIR coefficients. The coefficients have a gain of 1024 (10 bits) to avoid floating point arithmetic in Verilog, and the x-axis in the plot below has frequency normalized by the Nyquist frequency 24000 kHz. Thus, the 3 dB point cutoff is now 57 dB - lining up neatly with specified 120 and 260 Hz (0.005 and 0.011 normalized) cutoff frequencies.

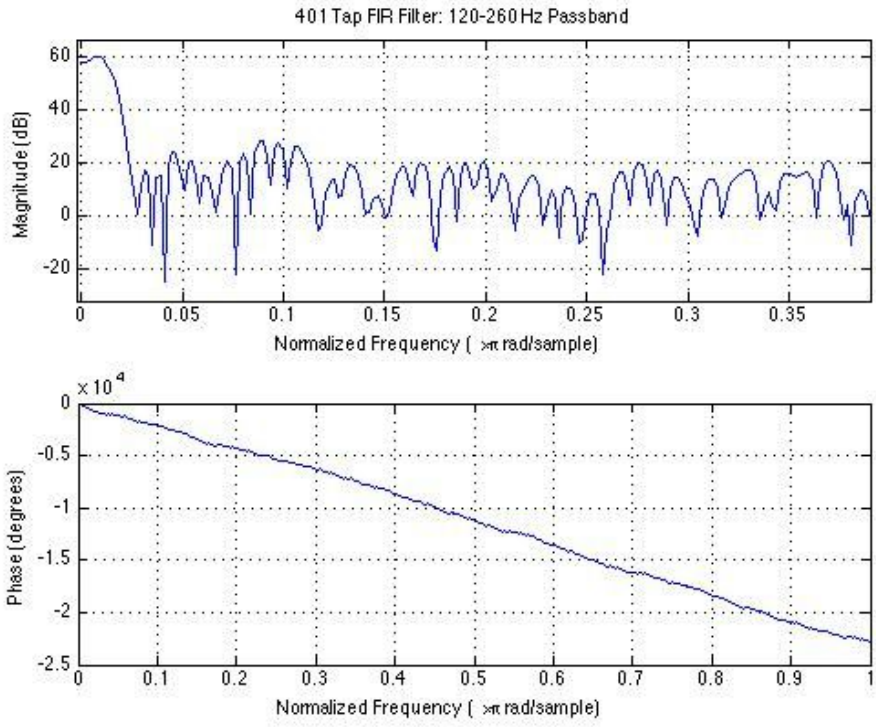


Fig. 4 Magnitude and Phase Data of 401-Tap FIR Filter: Passband 120-260 Hz

The implemented FIR filter design is based off of the 31-tap FIR filter described in 6.111 Lab 5; coefficients are stored in ROM and a circular buffer is used to store the 512 most recent audio inputs. However, there two main differences. Firstly, I use 401 taps rather than 31; 31 taps is not nearly enough data to produce a filter with sufficiently high precision at the low bass frequencies that we care about for music. Secondly, Lab 5 uses the 8 MSBs to perform the filtering and scales up by a factor of 10 bits (to avoid floating point arithmetic in Verilog) to obtain 18 bits. This technique does not produce audio of sufficient quality for our purposes, which I resolved by using a 28-bit accumulator and simply outputting the top 18 bits to avoid losing valuable data. A greater in depth explanation as to the design challenges I overcame can be found in my design experience section.

Once the frequency data has been obtained, the frequency module calls the equalizer module to perform the appropriate weighted addition of the signals and to store the user-selected weight of each of the individual filters. The user interface is configured such that the user can toggle the switches to hear either the untouched audio, the audio output of just one of the filters, or the audio output of the equalizer that sums up all of the weighted audio frequency bands. The user can increase/decrease the weight of a filter by toggling that filter's switch to "1" and hitting the right/left buttons on the labkit. The weight of each filter is 5 bits, providing the user with a range of 32 different possible weights for each of the 6 frequency bands. These weights are outputted to the labkit's hex display for ease of use. The 8 MSBs of each of the weighted frequency bands output by the equalizer submodule to the frequency module is passed directly to the mixer for volume control.

Transmission Module - Alex

On the second labkit, exact copies of all the major modules, except for the mixer and visual display modules, are instantiated. As such, the second labkit is capable of processing audio data to the same extent as that of the main labkit and outputting it through its own AC97 codec. Instead, however, the processed audio is sent to one of the user buses on the labkit for transmission to the primary labkit.

The top twelve bits of each stereo signal, along with the ready signal on the secondary labkit, are sent continuously to 25 of the pins on the user bus. Only 12 bits of each signal are sent as 12 bits contain more than enough data for high quality audio playback and additional bits would require more wiring without much added benefit. A ribbon cable takes these signals in parallel and carries them over to one of the buses on the primary

labkit. The primary labkit contains a complementary receiver module which monitors the ready signal sent over by the secondary labkit.

On every negative edge of the incoming ready signal it captures the signal data transmitted and synchronizes it with its own internal clock. In addition, it fills in the lower 6 bits of each stereo signal with the value of the highest order bit. This is done to reduce noise that occurs during sharp data transitions, especially when the signal changes signs from negative to positive. This data is then synchronously sent to the mixer module alongside the output of the system's own processing module in the form of two 18 bit stereo audio signals.

Mixer Module - Madeleine

The mixer module combines the two processed audio signals to create a single stereo audio output for the user to hear. The mixer take as input the two stereo audio signals (36 bits each) , 56 bits of frequency data (the 8 MSBs of audio data for each of the original 7 frequency ranges, though we only use 6 of them), and the selection controls from the user indicating how they wish to combine the signals. The mixer is capable of the following mixing combinations:

- output a selectable weighted combination of the two audio inputs
- output audio input 1
- output audio input 2
- output audio 1 with volume controlled by the volume of audio 2
- output audio 2 with volume controlled by the volume of audio 1
- output audio 2 with volume controlled by the bass frequency data of audio 1

The user can toggle between these different options using the switches on the labkit to connect these mixes to the output of the mixer and into the AC97 audio codec for output into the speakers.

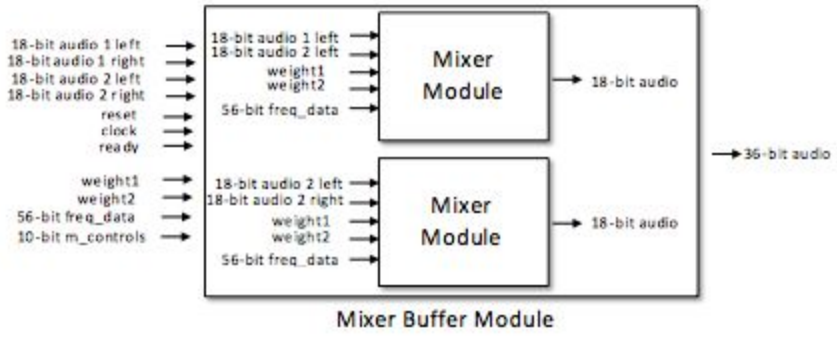


Fig. 5 Mixer Block Diagram

Shown above is a block diagram illustrating how the mixer is designed. The mixer has a high level module named "Mixer Buffer" that takes in the audio data for both the left and right audio signals. This module acts as a buffer, enabling the mixing of the left and right audio to occur in parallel. The buffer receives 72 bits of audio data and 56 bits of frequency data, initializes the two individual mixer modules, and passes the appropriate data into the individual left and right mixer modules. This design setup enables the labkit module to interface with just one mixing module for simplicity.

When outputting the two signals together, the mixer performs a weighted addition of the audio inputs such that the output volume remains constant without clipping. When the High Level FSM selects the mixer controls, the user can use the left and right buttons on the labkit to change the relative weighting of the audio 1 and audio 2 inputs. The weight1 and weight2 are each 5 bits buses and designed such that they always add up to 31. This enables the output of the mixer to remain a constant volume when mixing, while the relative weight of the two signals changes. To avoid clipping, the weighted addition of the signals is stored in a 23-bit bus (18 bits of audio + 5 bits of the weight), and the mixer simply outputs the 18 MSBs to the AC97 audio codec.

To play input audio 1 modulated to the volume of input audio 2, the audio 1 is multiplied by a weight assigned by the volume data of the audio 2 signal. This weight of the audio to be played is determined by the 5 MSBs of the audio 2 volume data (excluding the signed bit). To avoid extreme changes in volume, the weight has a minimum volume threshold such that the mixer always outputs music. Additionally, the weight only updates when it receives a positive audio sample, to simplify the calculation since we only care about magnitude as opposed to the output being positive/negative. Lastly, the weight only updates every 0.02 seconds (every 1024 ready pulses) to smooth out the volume transitions such that the music is still recognizable while being modulated by the opposite signal. This same approach is used for controlling the volume of audio 2 with the bass/volume data of audio 1.

Visual Display Module - Alex

The visual display module is instantiated only on the primary labkit. This module is responsible for lighting a custom LED display located on top of the labkit according to the volume of the music currently playing. It consists of the LED matrix, and other minor circuitry, on the labkit breadboard as well as a verilog module programmed into the FPGA that controls the LEDs.

The display itself is a simple array of 30 red LEDs. Each LED is connected on its positive leg to the 5 volt rail supplied by the labkit and a 110Ω resistor on its negative leg. This resistor, along with some wire, then connects the circuit to one of the pins on the labkits user bus. When the pin is driven digital low by the labkit, current is able to flow from the 5 volt rail to the pin illuminating the LED. When the pin is set high, little to no current flows in the circuit, turning off the LED. Each one of the 30 LEDs is thus able to be turned on or off independently as they are all wired in parallel to 30 separate pins on the user bus.

While many patterns and functionalities could be implemented using this LED matrix, and many more if more LEDs were added, the verilog module we created simply turns on or off LEDs based off the current volume of the audio signal being played. When the volume of the signal increases, LEDs are activated vertically along the first column on the left of the display, and then vertically along the next column and so on, culminating in the illumination of the LED on the top right of the display if the audio is near maximum volume.

Since the audio signal fluctuates between negative and positive and we do not just want the LEDs to flicker, the LEDs activated do not represent the current value of the audio signal. Instead samples are taken every few milliseconds. If the sample taken is positive, it is averaged with the last 31 samples and used to generate the output for the LED display. Otherwise the sample is disregarded, as adding negative numbers to the average would decrease its value when we are really just interested in the magnitude of the signal not whether it happens to be positive or negative on any given sample.

High Level FSM - Alex

Due to the limited number of user inputs available on the labkit and the large variety of customizable effects we decided to implement, a high level FSM was required in order to allow the user to switch control between the various modules. As implemented, this FSM allows the user to utilize almost the complete set of labkit buttons to edit each customizable module's settings, independently of the other modules' settings.

Within the FSM, the current state, i.e. which module is being modified, is being tracked. If the user wants to switch to control a different module, they simply press 3 to control the frequency module, 2 to control the time module, or 1 to control the mixer module thereby changing the state of the FSM. When transitioning to the frequency module, the current value of the switches immediately takes effect and passed as controls signals,

implementing whichever frequency filters or effects are switched on. For the time and mixer modules, the only the left and right buttons are immediately passed on to the module during a transition. If switches are activated or changed, the module is unaware until the user hits the enter button which then updates the modules control signals to match the current value of all switches.

This way of updating control signals, while slower and requiring additional user interaction, was chosen to avoid accidentally messing up user settings. For instance if a user was to switch from the frequency module to the mixer and both of them updated automatically, then the user could accidentally change which signal was being output or the method in which the signal was output depending on the frequency module settings prior to the transition. Requiring use of the enter button ensures that settings are only applied when the user activates them.

Design Experience - Madeleine

Designing and implementing a high quality audio processing and mixing system was an entirely new and educational experience for me. Prior to this class, I had very little knowledge of digital logic, and prior to the final project for this class, I had never worked with audio or signal processing. While I did not do the audio lab for Lab 5 of this class, I feel as though the labs leading up to the final project more than adequately prepared me to tackle this project, as they gave me the experience and knowledge of how to build and debug a complex digital system with timing constraints and carefully planned handshaking between the subcomponents in the system.

My main contribution to this project was the frequency processing of the audio signals to enable equalization, a task that took far more time and thought than initially anticipated. Outlined in the following paragraphs is the debugging strategy and method I took that enabled me to ensure that the frequency module met its desired specification.

I made the mistake of trying to write the Verilog FIR filter first before fine tuning the parameters in Matlab, and had to redo much of my implementation once I modified and decided upon a new design. Having never worked with filtering before, I spent the majority of my first couple weeks of the project testing and simulating the coefficients and FIR filtering in Matlab to fine tune the filters. Working with Matlab was an excellent way to debug and gain a better understanding of how digital filtering works; I wrote an FIR filter function in Matlab for testing purposes and generated the coefficients using the Matlab provided function "fir1". I began by passing sine waves through the filters to ensure that they were amplified/attenuated appropriately, and settled on a 401-tap FIR filter rather than a 31-tap filter. I made this design decision because it enabled me to have sufficient resolution at low frequencies without having to store too many samples in registers or needing to further pipeline the audio signal.

Once the Matlab filters performed as desired, I imported MP3 audio files into Matlab, applied the 6 FIR filters to it, and listened to their outputs. This way, I had a baseline with which I could check the validity of my Verilog implementation. Additionally, I wrote Python scripts to convert the Matlab FIR coefficients into Verilog so that I did not have to copy several thousand coefficients into Verilog by hand.

Lastly, I attempted to interface with the labkit's built-in flash memory, using the flash manager provided in the 6.111 tools section. One of our stretch goals was to store songs/sounds to memory and add them to the output via the mixer. I successfully wrote a module that could record to and play from the flash memory, but I did not finish

debugging this portion of the project to effectively integrate this functionality into our final system. While I did not succeed with this module, I gained valuable knowledge of how to work with Flash memory that I will take with me moving forward.

Overall, I thoroughly enjoyed my work on this project and learned relevant design, modeling, and debugging techniques that I will take with me in future engineering projects and classes, and I am glad I had the opportunity to design and build the FPGA DJ project for 6.111 this semester.

Design Experience - Alex

Like Madeleine, I had never actually worked with any sort of audio processing system prior to this project so learning about the various effects that are possible to implement in simple time and frequency based systems was very interesting. I discovered the wide variety of effects available to audio engineers in the processing of music and the results of combining these various effects in all sorts of combinations. In addition, I learned much about the technical challenges and requirements associated with storing and processing audio data.

While designing the FPGA DJ project I researched a large variety of potential audio effects to see what would be feasible and fun within a DJ system. During this time I was amazed at how many of the most popular effects, like echo, swell, flanger, chorus, and reverberation utilized simple signal processing techniques that had been covered in courses like 6.003. In particular, the power of feedback and feedforward paths made my portions of the project come together since they are the key component of all of the popular time audio effects listed above.

Implementing the delay required for audible time based effects was also very educational as it exposed me to memory interfaces. This exposure was required as storing the amount of audio data necessary for an audible delay needed much more memory than available in the simple registers we have typically used over the course of the labs. By utilizing the BRAM I was able to tap into a much larger storage pool and create large delays without sacrificing any audio quality. The IP Core generators provided with ISE also greatly simplified this process as I just had to specify the input output requirements and provide the proper connections, greatly abstracting any complicated aspects of indexing into large storage arrays.

Building the Visual Display Module was also a very fun experience. A primary interest of mine is analog circuitry so being able to incorporate even a little bit of outside circuitry into the project was a major goal of mine. I've also always had a desire to work with LED display matrices so getting to build a larger one where I had enough pins, as provided by the labkit, to individually control each LED was very enjoyable. Also, thanks to Gim's advice, I saved a great deal of time in this process by removing a large number of transistors that I originally thought I would need to activate each light.

One last area of the project that I played a major part in developing and found interesting was in the creation of the High Level FSM and general structure of the system. I recognized early on in the project that integration and control would be greatly

facilitated by the existence of a pre-existing infrastructure that we could drop finished modules into. To this end, my first major task, after getting basic audio playback, was to create such an infrastructure. This system level layout also greatly impacted the overall design and completion of the project by bringing our attention to particular connections where timing might be tricky or where we had not specified well enough how we wanted the system to behave. Though this module is likely the least visible to the user, I feel it was my greatest contribution for that very reason. Because of it, we had the framework and tools from which all other modules could build from and interact within in a seamless manner characteristic of any user-friendly electronic system.

General Issues Encountered - Alex and Madeleine

Throughout the implementation of the FPGA DJ system, we encountered a variety of challenges, some of them predictable and others entirely unforeseen. Namely, we encountered difficulties with communicating with the AC97 audio codec, needing to power cycle the labkit any time we compiled the project, and with ISE crashing randomly at times.

Much of our early work was done in parallel - once we designed the general structure of our project and agreed upon inputs and outputs, we generated skeleton files and worked entirely in parallel using GitHub to share our code. Alex worked on the time effects and Madeleine worked on the frequency effects, and while this worked well, adding additional layers once these effects were implemented could inexplicitly cause the AC97 audio codec to stop producing output. The exact reason for these glitches is unclear, especially when they occurred with modules like the Visual Display Module which is not actually in the signal path to the AC97. Utilizing continuously assigned wires instead of registers often solved these problems, but it would be good to know for sure why they occurred at all.

Another bizarre quirk of our program was that once it had a few main modules added, whenever we wished to program the labkit with the most recent version of our project, we had to power cycle the labkit. This anomaly began to occur about halfway into the development of our project, and took a while for us to recognize as initially we thought the issue was either a broken cable or an unforeseen bug that we introduced into our code. We did not confirm, but suspect the addition of BRAM and other memories into the project may have been the culprit. During the programming period, these memories may not be fully erased leading to problems during the initialization of the AC97 for instance.

Lastly, ISE crashed multiple times over the span of a week, causing me (Madeleine) to redo my work with the equalizer several times. Fortunately, we stored all of our code to GitHub, so, with the exception of a couple ill-timed crashes, our project was mostly protected and it was simply an exercise of rebuilding the project and re-generating the BRAM.

Potential Applications and Expansions

Our FPGA DJ project could be utilized for a variety of applications where signal processing is useful, as well as expanded to include many more effects. The most obvious and intended application of our FPGA DJ system is as a live audio processing a mixing tool for individuals to create fun, creative sounding sounds and music. We also recognize that one of the audio inputs to the system could be a microphone as opposed to a music source, allowing a user to sing over a song and process both their voice and the song through our system in a kind of karaoke-like fashion, mixing them together at the end with custom volume weights.

If a recorder was attached to the output rather than a speaker, then the user could save the processed and mixed signals, enabling the creation of cool replayable audio effects, mixes and mashups for future use. The system is not limited to working only with audio signals of course; any signal of a frequency below 24kHz could be filtered, echo'd and mixed according the user's desires and purposes. While we have not tried it, the system's filters and other features could lend itself to other types of data processing and output.

Lastly, the system could always be enhanced to handle more effects, such as flanger, chorus, phaser, distortion and many more that we have not even considered. In addition to more effects, the system could utilize the effects it already possesses in a more complex manner. For instance, the mixer could be upgraded to handle more than two signals, allowing for the parallel processing of many signals in preparation for final mixing and resulting in a more complex, layered final signal. Another upgrade would be allowing the user to simultaneously have multiple outputs from the system. For instance the user could record the complex mixed audio from both labkits, as well as the frequency data from all 6 filters independently for comparison. The possibilities are limited only by the upgrader's and user's imaginations.

Acknowledgements

We would like to sincerely thank and acknowledge all of the hours put into the course by Gim, Ariana, Miren and the LA's. Their incredible help and efforts were crucial in making this course and our project come together. Building this FPGA DJ system, and all of the labs in the course, has been an incredibly interesting and challenging experience and we would not have been able to do it without the patient and careful guidance provided by all of you. So again, thank you all for giving the course so much of your time and making it such a resounding success!