6.111 Revised Project Proposal: FPGA DJ

Alex Sloboda, Madeleine Waller

1. Project Overview

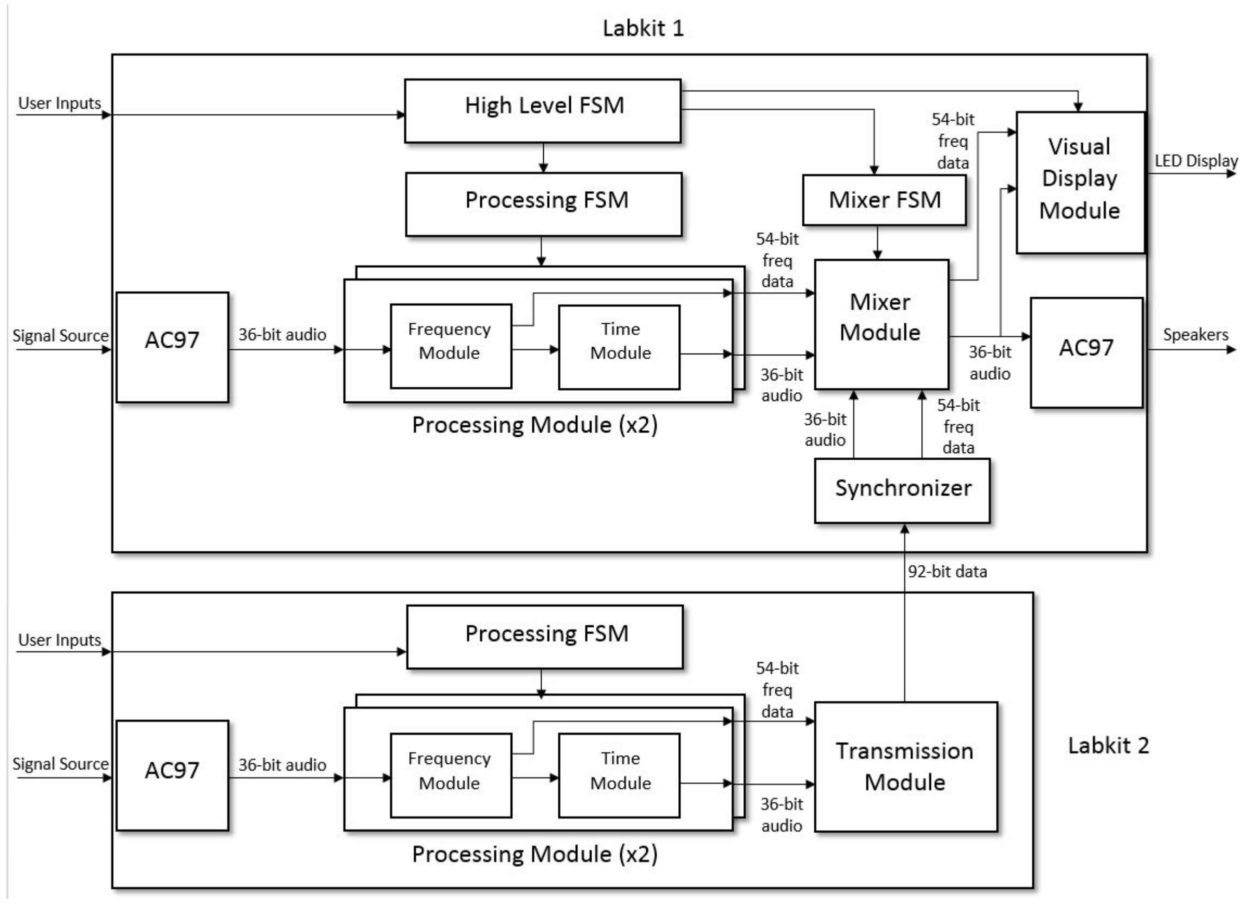
1.1 Summary

The FPGA DJ system is a music synthesizer capable of taking two audio signals, processing them in real time, and mixing them to produce a single handcrafted output consisting of surround sound and lights. The user will be able to hand select the processing effects of the input audio signals individually, on two separate labkits using the switches and buttons provided. The user can then custom mix the processed signals to produce a unique audio output to the surround sound speakers connected. We will have a visual output display consisting of a 6x5 grid of single color LEDs, which will provide volume and frequency information of the audio output of the synthesizer and make the system more aesthetically pleasing and interactive. Some core challenges we expect to encounter in the development of the FPGA DJ system include implementing and accessing BRAM on the labkit, properly tuning the filters for clean frequency effects, and correctly implementing the timing and pipelining of the time processing effects of the audio signals.

1.2 Motivation

Our project is exciting because we wanted to learn about and implement signal processing techniques, and because all of the effects and synthesis will happen in real time. This implementation enables the user to simply plug in their audio input(s), select the effects, and hear the output of the synthesizer via stereo speakers with minimal delay. The challenge lies in pipelining the filtering and processing techniques to enable this instantaneous output. Lastly, 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 hundreds of different combinations, adding a great layer of complexity and potential to the user experience and the system itself.

2.1 High Level Block Diagram



2.2 Design Implementation

Our FPGA DJ system will consist of two parallel processing modules acting upon the two independent, audio signals which then will be sent, post-process, to a central mixer in order to be output to a pair of speakers and a light display. The central mixer, one of the processing modules and a light controller will be located on one labkit, while the other labkit will contain only the other processing module and a transmission module responsible for sending its final output to the mixer labkit. The audio signals will be input into the system using the AC97 audio codec.

Within the processing module, there will be an identical pair of two major submodules for each channel, a filtering module and a time processing module. The filtering module will be able to pass the input signal, unfiltered, to the time module, or carry out filtering of the signal to result in a custom weighted combination of 7 different frequency bands. In addition, the filtering module will send out information on the volume of each frequency band for use in the mixer module regardless of whether the output of the filtering module is filtered or not.

The time processing module will take the output signal of the filtering module and perform processing on the frequency altered signal, primarily through the use of feedforward and feedback implemented using registers. As such, the time processing module will be able to create effects such as swells, echos, reverberations, and chorus'. It, like the frequency module, will also be capable of passing through its input signal without having it undergo any processing. Whichever effect it implements will be carried out and the output will be a processed audio signal.

With the processing complete, the processed audio signals from both labkits will then be sent to a mixer module. In order to get the output from the labkit not containing the mixer, wires will carry the audio information and frequency data to the mixer labkit from the secondary labkit. The mixer will then allow for one final level of control, as it will allow for effects such as a weighted sum of the audio signals or the volume control of one signal by a frequency of the other signal. Following this mixing, the audio will then be directly sent to the AC97 once more and played over the speakers. The output will also be sent to the visual display module.

The visual display module will interface with LEDs on the top of the labkit, turning each one on or off individually to create effects such as a frequency spectrum analyzer. The audio processing and filtering is the main focus of the project so at this time the full potential of the visual display module has yet to be completely defined.

In order for the user to control the audio output of our DJ system, multiple FSM's will be used and toggled between. Within the processing module, two FSM's will control the frequency and time modules respectively. These themselves will be controlled by a higher level FSM responsible for running the processing modules, mixer module and light display module. The user will be able to toggle control between these submodules using the higher level FSM, allowing for a large range of independent custom settings to be created across the modules.

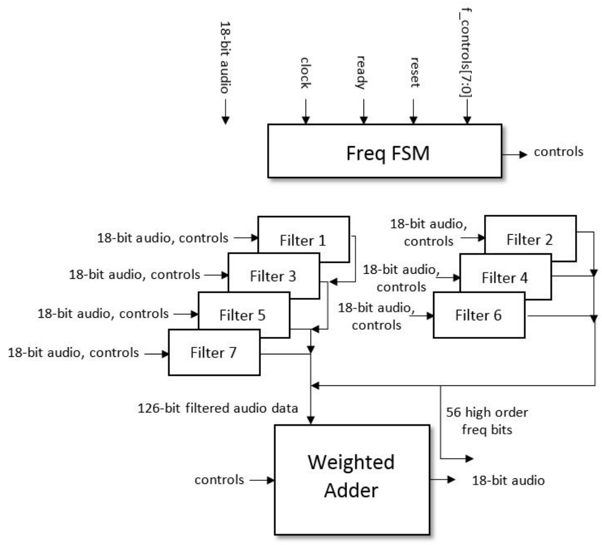
3. Module Descriptions

3.1 AC97 Module

The AC97 module serves as both the initial input and final output of our system. Utilizing a modified version of the verilog provided in lab5, the AC97 samples a stereo signal from the RCA inputs of the labkit and outputs two 18 bit audio signals representing the left and right channels. These signals are presented to our system at a rate of 48 kHz for processing. Once all of our processing has been completed, the AC97 is sent our two finished 18 bit audio signals to be output to a pair of speakers.

3.2 Processing Module - Frequency Module

Shown below is a lower level block diagram of the frequency processing module - a module that is responsible for the frequency effects performed on the audio inputs. It has 3 main components: the Frequency FSM, individual filters, and a weighted adder.



Low Level Block Diagram of Frequency Module

3.2.1 Filter Module: Frequency FSM

The frequency FSM module is an FSM that receives information specifying which frequencies are to be amplified or attenuated and enables the appropriate filter module to perform the amplification/attenuation on the specified frequency range. The frequency FSM module will take as input the 18 bit audio input signal and will return as output a fully processed 18 bit audio signal along with the 56 bits of frequency data to be be passed as input into the synthesizer.

The frequency FSM module will call seven filter modules to perform the filtering: one instance of a filter module will be capable of amplifying one particular frequency range and attenuating all the rest. The frequency FSM module can then add the various signals as instructed by passing the appropriate information to a weighted adder, e.g. adding the bass frequency range to the original signal to amplify the bass without changing the rest of the audio signal.

Filter Module: Individual filter3.2.2

The individual filter module is a 31-tap FIR filter that will amplify a specific frequency range while attenuating all frequencies outside of this range. This will enable us to boost certain frequencies, e.g. the bass or treble in a song. The individual filter module will be called by the frequency FSM module, and will take an 18 bit audio signal in the time domain as input at each clock cycle and return an 18 bit audio signal in the time domain as output. To perform the necessary calculations, the filter module will store the last 31 18-bit samples. The FIR filter calculation entails the weighted addition of the 31 most recent audio samples, meaning 31 multiplications for each computation. This is too many multiplications for one clock cycle, so we pipeline the computation over 31 cycles of a 27 MHz clock. The output will contain only the elements of the audio signal that yield a certain frequency range, attenuating all other frequency ranges.

The filter module will be instantiated 7 times to enable parallel processing of signals - one for each frequency range we wish to process, and each instantiation will contain the coefficients necessary to perform the filtering computation. These coefficients need only be computed once at compile time, and do not need to be updated when the system is running. The specific frequency ranges will need to be fine tuned as we test the system to cleanly process a wide variety of songs. To fine tune the filters, we will use Matlab to compute the necessary coefficients to be entered into Verilog.

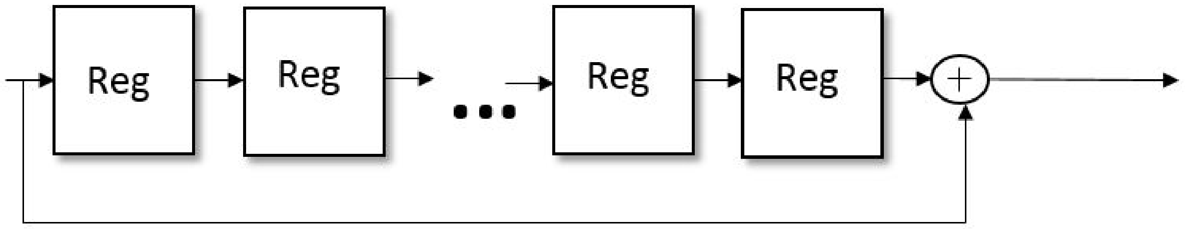
We have chosen to use filters to obtain frequency information instead of performing an FFT and an IFFT because the Xilinx's CoreGen is currently unable to produce the Verilog code for FFTs and IFFTs, and the IFFT would be too difficult for us to generate on our own.

Filter Module: 3.2.3 Weighted Addition

The weighted addition module is a helper module called by the filter frequency FSM module whose purpose is to add the various audio signals and produce a final signal as output. This module will take as an input parameter the number of audio signals to add, the 18 bit audio signals as inputs, and the weight of each input, and will output a single 18-bit audio signal. For instance, this module can perform the addition of the original audio input to several processed inputs (e.g. low bass or mid range) when called by the frequency FSM module, or to add two individual processed signals weighted as specified by the mixer.

3.3 Processing Module - Time Module

The time module take as its input the audio signal output of the filter module. With this audio data, several parallel effect modules will act upon the signal, feeding their resulting audio streams into a single mux at the end of the module. Utilizing the processing module FSM, this mux will then decide which effect will be sent on to the mixer module.



Feed-forward “Swell” Effect Implementation

Unlike the filter module, the effects carried out in the time module will be time based and implemented utilizing feedback/feedforward paths as depicted above. While the final list of effects that will exist has not been pinned down, we intend to implement at minimum a chorus effect, reverberation, echo and reverse echo or swelling. All of these effects listed depend on adding attenuated versions of the original signal to itself, but with some form of time delay in the positive, negative or both directions.

3.4 Processing Module - FSM

The processing module FSM will be a simple state machine that will allow for the user to toggle through the various effects implemented in our project and combine them to create more complex, interesting sounds. The processing module will take user inputs and toggle between controlling the filter module and time module. When in control of the time module, the user will be able to either rotate through the various effects implemented or select a specific effect and immediately set it as the current time based effect.

When in control of the filter module the user will be interfacing with the frequency FSM module as described above. Since the filter module is more complex, it may not be possible to instantaneously achieve the desired output. For instance, a signal containing a weighted the sum of three frequency ranges will require the user to select each individual frequency range and either amplify or attenuate it appropriately to obtain their desired output rather than simply toggling that specific output. Information on which frequency is being interfaced with and what level of attenuation/amplification is being performed, along with other to be determined system information, will be output to the labkits onboard LEDs for user convenience.

3.5 Mixer Module + Mixer FSM

The mixer module will be responsible for combining the two processed audio signals to create a single stereo audio output for the user to hear. The mixer FSM will take as input the two 18 bit audio signals for stereo (36 bits total), 56 bits of frequency data (the 8 MSBs of audio data for each of the 7 frequency ranges), along with selection inputs from the user regarding how they wish to combine the signals. The FSM will use the selection to establish which helper module to call to perform the mixing of the signals. One helper module will be performing a weighted addition as input, and will simply be an instantiation of the weighted addition module described in section 3.2.3. Another helper module will be called to modulate the volume of one of the audio signals to a specified frequency range of the second audio input by finding the average volume of the second signal in the specified frequency range and using negative feedback to adjust the volume of the first signal accordingly. If modulating the volume of one signal to the beat of the other is selected, then both helper modules will be called - one to process the individual signal and then one to add the two signals for the final output into the speakers.

3.6 Visual Display Module

The visual display module will be responsible for interfacing with our external LED display matrix. It will take as input the frequency data being used by the mixer module, the audio output of the mixer module, and user inputs. Based off of the user inputs, it will be capable of switching between various pre-made display modes, such as a frequency spectrum analyzer, and activating lights accordingly for an aesthetically pleasing DJ experience.

3.7 Transmission Module + Synchronizer

The transmission module and synchronizer will be simple modules implemented on labkit 2 and labkit 1 respectively. On labkit 2 the transmission module will take the output of labkit 2’s processing module and output it via the digital bus’s on the labkit through 84 wires so that labkit 1 can use it in its mixer module. In order to properly receive this data, labkit two will contain a synchronizer for all of the 84 signals to avoid any glitches or metastable inputs. The synchronizer will then send this data to the mixer module in the same format as the processing module.

3.8 High Level FSM

The high level FSM will be instantiated on labkit and will help interface with all of the FSM’s controlling the individual modules. It will be a far simpler FSM, simply toggling control between the module FSMs and passing them the current set of user inputs. This FSM is required due to the limited number of total inputs available to the user and will allow for a much wider range of control over the audio and mixing effects implemented. Labkit 2 will not possess this module, instead user inputs on labkit 2 will be directly sent to the processing module without any interceding module.

4. Equipment Needed
In order to implement our project as described above we will require a moderate amount of materials. As described, our project will be implemented upon two labkits which will already be in the lab. In addition we will require speakers and audio sources for sourcing and playing our sound, these are also already found in the lab or readily exist in the form of phones respectively. In order to get stereo audio from our music sources to the labkits, we will require two 3.5mm to RCA adaptors which we have already purchased online.

What remains to be found are primarily related to the LED display we intend to create. In order to create a 6x5 matrix of LEDs we will require 30 blue LEDs plus extra to handle bad parts or mishaps. To properly activate the LEDs using the digital outputs available from our labkits we will also need 30 discrete MOSFETs acting as switches, plus extra to handle bad parts and mishaps. To ensure we don't exceed the power limits of the labkits 3.3V rail or damage the LEDs we will also need 30 500 ohm resistors, plus extras to handle bad parts and mishaps.

Finally we will require a decent amount of wiring in order to connect all of our LEDs, MOSFETs and voltage rails and to connect the second labkit to the mixer labkit. In order to ease the connection of this wiring to the labkits and breadboard it would also be good to have headers and corresponding multi-wire cables if available.

1. Labkit x2
2. Speakers (stereo pair)
3. Sound source (phones, laptops, etc) x2
4. 3.5mm to RCA stereo adaptor x2
5. Blue LEDs x40
6. Discrete MOSFETs (to control LEDs) x40
7. 500 ohm resistors x40
8. Wires

5. Timeline

Shown below is a Gant chart indicating our proposed timeline to implement our project on time and to specification. We divided the tasks as follows:

Alex Sloboda:

1. Obtain basic audio in, stereo audio output
2. Build FSM to enable user to select desired processing effects
3. Implement Time Based Processing Modules
4. Build and Implement LED display

Madeleine Waller:

1. Research frequency filtering techniques and choose implementation
2. Build frequency filtering module prototype in Matlab
3. Implement filters and frequency based modules in Verilog
4. Implement mixer module

We left a significant amount of time for implementing the frequency filter in Verilog because Xilinx ISE does not currently have a working IP Wizard to build an FFT and IFFT, so we must build filters for specific frequency ranges and perform our signal processing entirely in the time domain instead of performing it in both the time and frequency domains.

Shown below is a Gant chart of our proposed timeline. Alex is responsible for the purple tasks, Madeleine is responsible for the blue tasks, and red tasks are a group effort. We will be debugging as we go along, testing each of the modules independently and developing simulations when relevant. This will simplify the final debugging process when we merge the subsystems to build our final FPGA DJ system.

	Week of 10/26	**Week of 11/2**	**Week of 11/9**	**Week of 11/16**	**Week of 11/23 (Thanksgiving)**	**Week of 11/30**
Audio In/ Stereo Audio Out	🟪					
Research Filtering Techniques		🟦				
Design Filtering Techniques in Matlab		🟦				
Implement and Debug Frequency Filters in Verilog		🟦	🟦			
Implement and Debug Time Based Processing Modules		🟪	🟪	🟪		
FSM to select effects (ongoing)		🟪	🟪	🟪	🟪	
ImplementFrequency Based Processing Modules			🟦	🟦		

Build LED Display				🟪		
Implement and Debug Mixer Module				🟦	🟦	
Implement LED Display Control Module				🟪	🟪	
Stretch Goals					🟥	🟥
Final Debugging					🟥	🟥

6. Conclusion

The FPGA DJ system is an exciting project because it enables the user to mix music in real time, and has an added layer of complexity due to the various processing techniques and pipelining that we must perform to enable the user to select and hear the synthesized audio in real time. If we accomplish our immediate goals discussed in this proposal, then we will take the project a step further design and implement more complex time and frequency processing modules (such as a phaser or a flanger), enabling the user to add pre-recorded sounds to the audio output from memory (e.g. drums), and creating a more complex LED display than currently proposed.