Hardware Parametric Equalizer Proposal

Background Motivation

Audio equalizers are important tools for adjusting the gains of different frequencies so that the sound output is more desirable. Typically, they are used to help offset unique differences in the frequency responses of speakers in order to more accurately replicate the original sound that was recorded.

A parametric equalizer provides an important additional benefit: it allows the user to actually select the size and center of the frequency band. This provides very fine control over the audio output; the user can build a filter of nearly any shape. Most parametric equalizers are implemented in software, but it should be possible to implement an equalizer device in hardware for lower latency processing.

Implementation Overview

A high level system flowchart is available at the end of the document.

User Interface

Display

The VGA output will show the current gain curve and display dialogs for the two input modules described below. If additional processing space is available on the chip, overlapping waveforms of the input and output audio streams as well as an FFT of the audio data will be showed on the screen as well.

To accomplish this, the screen will be divided into several sections, with a module responsible for updating each section. For example, the gain curve display will module will have a input signal that is pulsed when the gain curve has changed. When this occurs, the module will read the new gain curve from memory and compute the appropriate graph. A different module will examine the magnitude of each sample of the incoming audio stream and compute a graph based on that. The outputs of these modules will then be combined together to form the entire display.

Input Methods

There will be two methods for entering a gain curve. The first allows the user to create precise filters based on entering numerical parameters. After switching to this input method, the user will presented with a flat gain curve. The curve can be manipulated by adding "gain modifiers:" the user enters a center frequency, a bandwidth, a curve shape, and a new desired gain for that center frequency. The system then uses this information to modify the existing gain curve. As the user adds more gain modifiers, the resulting overall gain curve will become increasingly complex. When this method is chosen, a state machine will control the flow of the input data. The user will press a button on the Labkit to add a new gain modifier and then use the up and down buttons to key in values for the frequency center, bandwidth, and gain. The user will then press another button to confirm the update.

The second input method provides a holistic way of modifying the audio data: the user will use the mouse to draw a gain curve on the screen. Each atomic horizontal movement of the mouse to the

right will increase the index of the gain curve interval and a block will use the change in the vertical position to increase or decrease gain for the new interval relative to the last.

The user will be able to switch between the two input methods; the gain curve will be reset each time they switch the input method.

Signal Manipulation

This gain curve that the user interface produces is effectively a FIR filter of some arbitrary shape in the frequency domain. That is, for each frequency interval, there is an associated gain value. The signal processing block of the proposed system will use this data to equalize the incoming audio stream.

There are two main approaches for applying a gain curve or FIR filter to an audio stream. The first method would involve converting the gain curve to a time-domain filter and convolve it with the incoming audio stream. This method takes the classic approach to filter application; applying the filter in the time domain is computationally inexpensive.

However, the process of building the FIR filter is fairly complex. There are several algorithms for filter construction that can be implemented in hardware including the Parks-McClellan algorithm, which efficiently approximates a filter. While some work has been done to create various implementations of the Parks-McClellan algorithm on FPGAs,[1] [2] the implementation of the algorithm capable of performing the required tasks would be very difficult.

The second method for applying the gain curve to the audio stream is to take an n-point FFT of the incoming audio data, normalize and multiply each point by its associated value on the gain curve, and take an inverse FFT of the data, sending it to the AC'97 output port. Though there is a loss of phase information, this method is far easier to implement than the first method, especially given that Xilinx has FFT and IFFT cores available for implementation. This method is a bit more expensive than the other method in terms of the resources and time required to implement the filter itself. However, due to the simplicity of this design, it will be chosen for this system.

The Xilinx FFT core[3] is highly customizable and a single instantiation can perform both an FFT and an IFFT depending on the setting of a control bit. The digitized audio stream will enter an instantiation of this core (in order to have enough resolution, the FFT will likely need to be 2048 points, but simulation tests should be performed to determine an acceptable value).

The FFT will store its output data to a n-address, 8-bit RAM where n is the number of FFT points. Next, the parallel multiplication network[4] will multiply each point of the FFT with its associated value on the gain curve. Each of the n operations will therefore require two memory reads (one for the gain curve value and one for the FFT value) and a multiplication. A significant amount of chip space will likely be dedicated to this module (the FFTs will be configured for minimal usage of the on-chip multipliers). Finally, when all of the data is ready, FFT block's inversion control bit is switched so that the module reads the newly computed data from memory, takes an IFFT of it and sends the output to the AC'97 codec.

The FPGA's clock frequency is significantly higher than the audio sampling frequency, so it should be possible to implement a design that performs all of the processing between audio samples. If it isn't, a more complex architecture that downsamples the incoming stream will be necessary (as well as a reconstruction filter on the output).

[1] http://ijict.org/index.php/ijoat/article/download/generic-fir-filter/fpga
[2] http://etd.ohiolink.edu/send-pdf.cgi/Buxa%20Peter.pdf?wright1183475958
[3] http://www.xilinx.com/products/ipcenter/FFT.htm
[4] http://www.xilinx.com/products/ipcenter/multiplier.htm

Memories

In addition to the internal memories that the FFT cores will create, two main memories will be used for storing data. The first will store the output of the FFT. When the FFT has finished, it should send a signal to a state machine, which will start the parallel multiplier network. This network will read from the second RAM, which stores gain curve data and multiply each entry of the FFT with its corresponding entry of the gain curve data. When the multiplier module has finished this task, it will send a signal back to the same state machine, which will then run the FFT in inverse mode and that data will be sent out to the output port.

External Interfaces

Almost all of the required hardware is available on the Labkit. VGA output will go to a display, mouse input will come from a PS/2 mouse, and the interface for adding gain modifiers will probably use either the buttons on the Labkit or a PS/2 keyboard.

However, some additional hardware may be necessary for getting the audio output from some sound source. If the RCA connectors on the Labkit support line-level inputs, no additional hardware will be necessary. The LM4550 has a line-level input on the chip, but it is unclear whether it is connected on the Labkit.

If no line-level input is available, one option is to convert the line-level audio output from a stereo or a computer to the mic-level input required for the AC'97 input audio outputs from e.g. a computer to the mic-level input required for the AC'97 chip on the Labkit. This can be done with a voltage divider using a 10K ohm resistor and a 100 ohm resistor.

The other option is to interface with an external ADC. This would require significant extra work and would not guarantee good sound quality unless the ADC is specifically designed for audio processing, so it is not an optimal solution.

Design Plan

Week of November 8

This week will be mainly for algorithm research and design. A working simulation of the equalizer's audio processing unit will be developed in Matlab to prove its functionality. It will use an 8-bit PCM file as an input and a generate an equalized 8-bit PCM on its output based on a given gain curve. An efficient parallel multiplier network algorithm will be designed and tested on the FPGA. The main user interface algorithms will also be designed: the process of updating the gain curve given a new gain modifier is somewhat mathematically complex and will require significant design and testing time.

Week of November 15

The audio processing block will be implemented. Using a preset gain curve stored in memory, the processing block will equalize an incoming audio stream and send it to the AC'97 output. If it becomes evident that the FPGA won't have enough room to store the data in SRAM, an interface for the ZBT RAM will be created and tested.

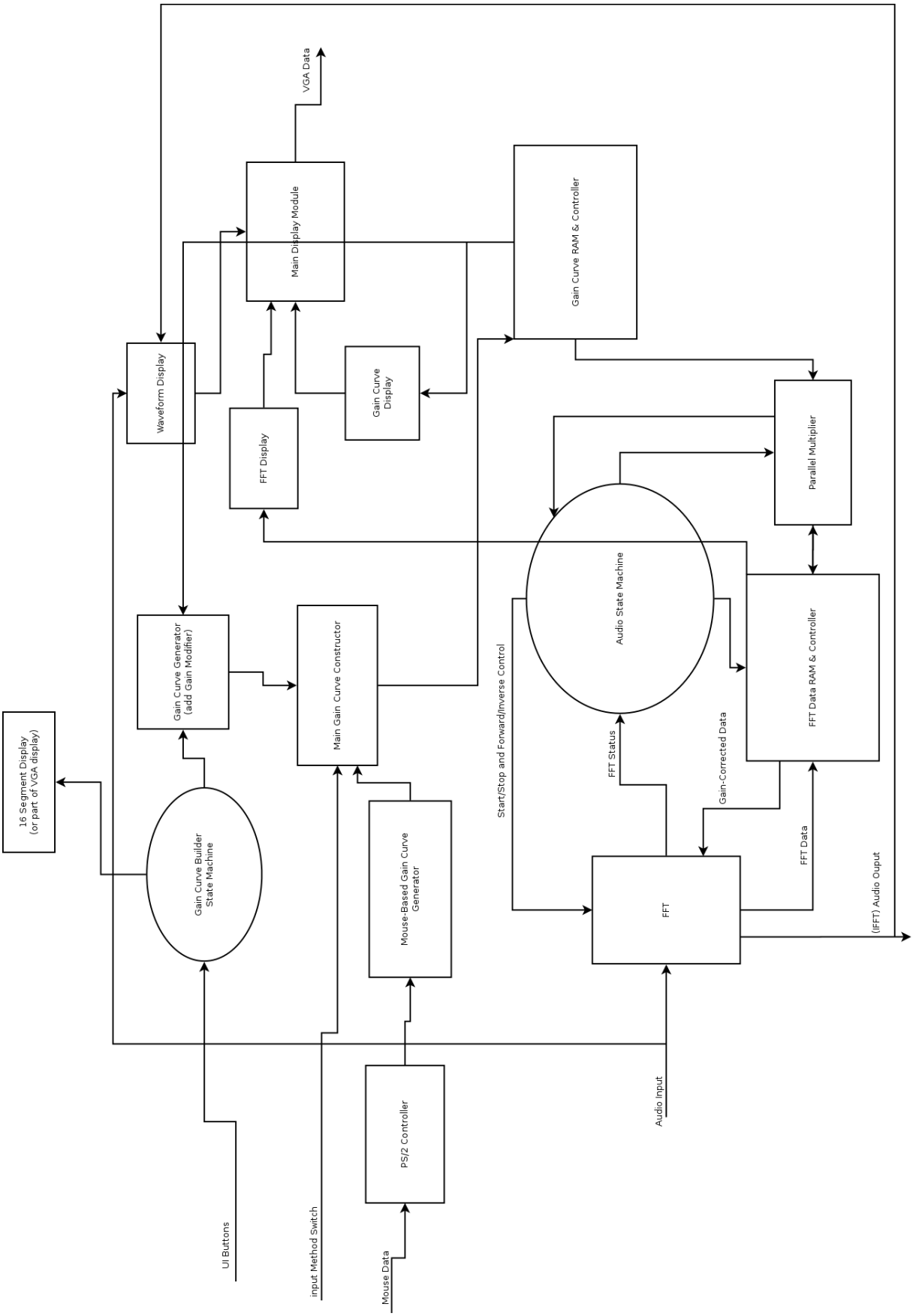
Initial implementations of the various smaller user interface modules such as the VGA driver and gain curve display will be completed. Also, the mathematical algorithms for updating the gain curve based on a gain modifier input will be implemented and tested.

Weeks of November 22 and 29

The two methods for designing new gain curves will be completed and the user interface will be fully implemented. If there is additional time, the display will be improved to include additional the input and output audio waveforms as well as live graph of the FFT.

Additional Work

If time permits, a few extra modules could be added to the proposed system to create a device capable of automatic equalization. The system would be extended to use four speakers and a microphone would listen to the output of the speakers. If the frequency response of the microphone is known, then the true frequency response of the four speakers can be obtained and a gain curve can be automatically built to correct against the abnormalities in each speaker.


VGA Data
Main Display Module
Gain Curve RAM & Controller
Waveform Display
Gain Curve Display
FFT Display
Parallel Multiplier
Audio State Machine
Gain Curve Generator (add Gain Modifier)
Main Gain Curve Constructor
FFT Data RAM & Controller
16 Segment Display (or part of VGA display)
Start/Stop and Forward/Inverse Control
FFT Status
Gain-Corrected Data
Gain Curve Builder State Machine
Mouse-Based Gain Curve Generator
FFT
FFT Data
(IFFT) Audio Ouput
PS/2 Controller
Audio Input
UI Buttons
input Method Switch
Mouse Data