The Spatial Digital Equalizer

Priya Kikani & Alexander Sludds

Abstract:

While the sculpted chambers of Carnegie Hall offer an amazing musical experience, small rooms muffle and distort the grandeur of any listening experience. The acoustical properties of the room, such as its size and the material composition of its walls, directly impact the musical experience. Our goal is to demonstrate that by using spatial acoustical characterization and digital signal processing to adaptively equalize recorded music, sound quality will become independent of the environment's intrinsic acoustical characteristics. First, a frequency sweep will be transmitted into the room and the reflected signal carrying acoustical information about the room will be collected. An FPGA will be utilized to generate a transfer function of the space. This data will be processed--also using the FPGA--to calibrate audio filters. These filters will pre-process the music with frequency-dependent compensation for the specific acoustic characteristics of the room. This project will immediately provide users with concert-hall-quality music within virtually any space they choose.

External Components

This project requires the use of speakers and a microphone for transmitting and recording data. The project will use the onboard microphone and speakers already in our possession.

Functional Specification

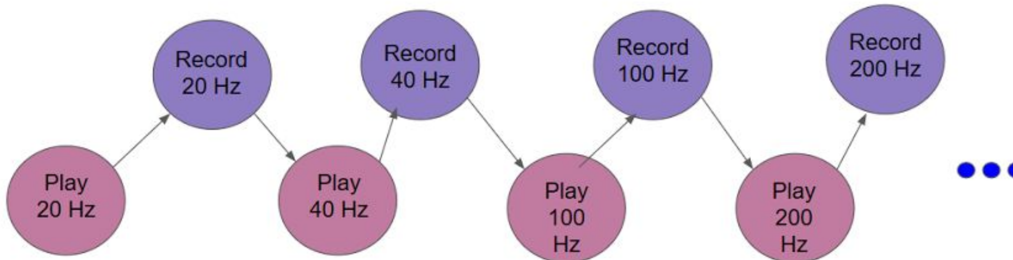
The block diagram is given here:

Module Description

Module 1: Record/Playback FSM

The first objective is to generate a transfer function of the room in which the music is being played. Because different frequencies of music will present different responses in a given room, a transfer function of the room will be generated by sending multiple tones of different frequencies and recording the reflected signal.

The first module is a linear finite state machine, graphic given below:



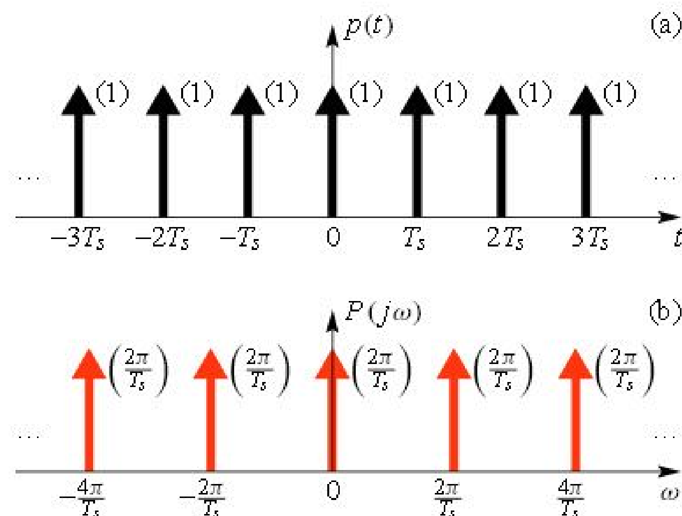
State 1: Play XX Hz

Inputs: none

Outputs: tones of defined frequencies, 16 bit audio

By Priya Kikani

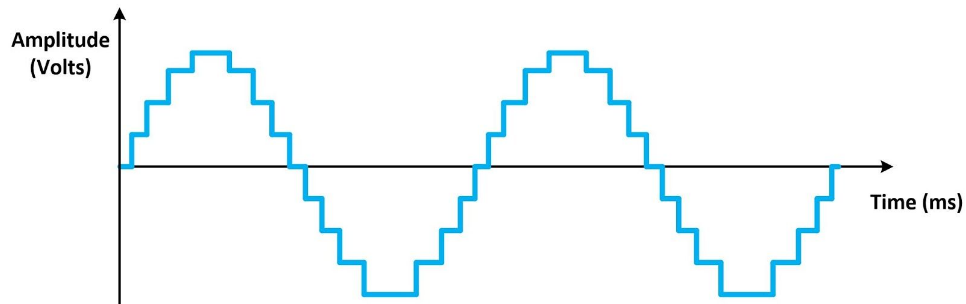
The accepted standard for audible frequencies is 20 to 20,000 Hz; furthermore, the human ear perceives audio on a logarithmic scale. As a result of those two facts, the tone generator will play 2 second bursts of sound at the following frequencies: 20 Hz, 100 Hz, 200 Hz, 1,000 Hz, 2,000 Hz, 10,000 Hz, and 20,000 Hz, thereby generating an impulse train in the time domain. As given in Figure 1, an impulse train in the time domain presents itself as an impulse train in the frequency domain.



Periodic impulse train: (a) Time-domain signal $p(t)$; (b) Fourier transform $P(j\omega)$. Regular spacing in the frequency-domain is $\omega_s = 2\pi/T_s$

Figure 1: Impulse train in time domain yields impulse train in frequency domain

In order to implement this tone generator, an approach similar to the Lab 5 will be used, with a 16 bit sine wave of a specific frequency being sent to the onboard DAC, example given below.



This module will be tested by listening for audio output as well as verifying the specific frequency being played with an oscilloscope.

State 2: Record XX Hz
Input: reflected signal from room, 16 bit audio
Output: reflected signals in memory, 16 bit audio in memory
By Priya Kikani

The signals outputted from the tone generator will travel across the room and reflect off the walls of the room. Because those reflected signals will carry information about the room, the objective of this module will be to record those reflected signals in memory for later analysis.

In order to allow time for the signals to propagate and return in a room, about 20 seconds total of audio will need to be stored in memory. In order to have enough room in memory, the signals will be sampled at a specific rate that is a function of its frequency and stored in an individual mybram module. For example, the 20 Hz signal only needs to be sampled at at least a 40 Hz sampling rate, while the 20 kHz sample will require a much higher rate; thus the lower frequency components can occupy a much smaller proportion of FPGA memory. In Verilog, this module will require a parameterized sampling rate that is passed into the recorder state that writes *to_ac_data97* based on a specific counter value. The recorder state will call on the mybram module within it. This overall module will be tested by implementing a basic playback function and making sure that the recorded signal loosely matches the original signal.

After this initial implementation, the signals will be recorded using a microphone set at some fixed distance away. This is to ensure that the later signal processing optimization will optimize for room geometry as the signals will have time to interact with the room before being recorded.

If necessary, other avenues for storing memory, such as ZBT memory, will be explored. Given that the overall calibration of the room does not require real time analysis, this record and playback FSM can occur over multiple clock cycles.

Module 2: Peak Detection and Frequency Comparator
Input: reflected signals in memory, 16 bit read from memory
Output: frequency response of room, 32 bit filter
By Priya Kikani

In order to determine the frequency response of the room, the relative amplitude of the returning impulse at each frequency will be compared. For example, if the returning impulse from a low frequency component is attenuated in comparison to the amplitude of the higher frequency component, the room can be approximated as a high pass filter. It is expected that tones at all frequencies will be attenuated by some amount due to the lossy nature of the room and recording equipment; however, we are only interested in the relative attenuation.

To implement this in Verilog, the first step will be to read each tone from memory. Next, the signal will be squared. One important fact to note is that this calculation can occur across multiple clock cycles because this is the calibration stage. Then, the moving average of each tone will be computed to generate filter coefficients for each frequency band. Designing for a 32 bit output allows for high resolution computation of the filter coefficients as well as consistent inputs to the Digital Signal Processing module.

This module will be tested with ModelSim. The inputs to the test bench will be artificially generated tones of various frequencies and amplitudes, acting as if they were read from memory. The output will be stored in a file that can be imported into MATLAB and graphed to examine the nature of the filter. The ideal output will look something like below**:**

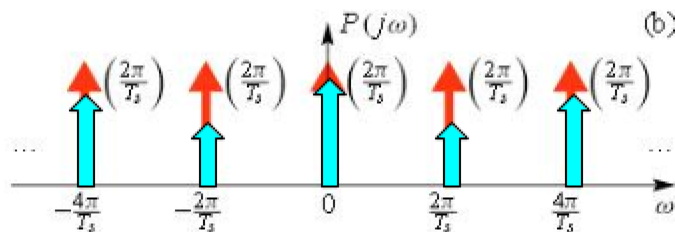


Figure 2: Demonstration of ideal filter output

In Figure 2, the red arrows represent the original frequency components of the signal, and the blue arrows represent the corresponding frequency adjustments to produce a transfer function of the room and the output of this module.

Module 4: Music Input

Input: hardware input from microphone over 3.5mm jack
Output: 20 bits of left_data and 20 bits of right_data
By Alex Sludds

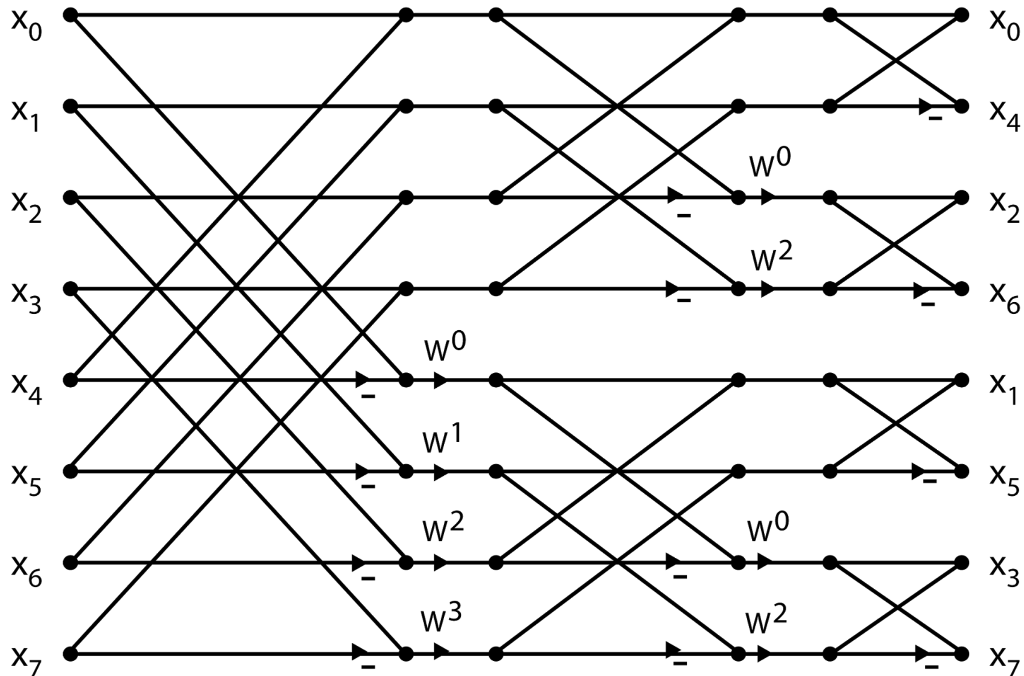
In this module we shall be making use of the readily available code from lab 5 for how the ac97 module functions. This module takes the above listed inputs and returns several outputs, most importantly of which are the 20 bits of left channel data and the 20 bits of right channel data.

This data is then passed onto an instance of the FFT module.

Module 5: FFT

Input: s_axis_data_tvalid, s_axis_data_tdata, s_axis_data_tlast , S_Axis_Config_Data [7:0], S_Axis_Config_Tvalid, M_Axis_Data_TReady, CLK
Output: m_axis_data_tdata, m_axis_data_tuser, m_axis_data_tlast, M_Axis_Data[63:0], s_axis_data_tready
By Alex Sludds

This IP Core module was created by Xilinx and relies upon what is called a "Cooley-Turkey" algorithm in order to create the DTFT (Discrete Time Fourier Transform). The Cooley-Turkey algorithm relies upon a principles called 2^n radix. With an input with 2^n bits we can procedurally add the bits together with a phase delay in order to create an output that represents the DTFT of the input bits.



Our implementation of this module will get the 12 most significant bits of data from a channel of the ac97 output and pass it into the tdata input.

S_axis_data is an AXI channel that carries two twos complement floating-point padded signals representing the real and imaginary parts of the signal respectively. It also includes s_axis_data_tvalid, a signal notifying the FFT module that data is available to be provided, s_axis_data_tlast, a signal asserted at the end of a frame of data, and s_axis_data_tready, an output from the FFT module used to signal that it is ready to accept more data.

When data is ready at the output of the FFT module we note that m_axis_data_tready will go high and that data can be taken from m_axis_data_tdata. It is within this output that we find our two real and imaginary data outputs.

Please note that documentation for the FFT IP Core can be found here:
https://www.xilinx.com/support/documentation/ip_documentation/xfft/v9_0/pg109-xfft.pdf

Module 6: DSP

Input: Data from the FFT of the music and the compensation filter.
Output: Signal to be passed back to ac97 module representing music to be played 14 bits
By Alex Sludds

The scalable section of our project is to create a DSP module that can be customized to do progressively more interesting forms of filtering. For example, a most simple DSP that can be performed is to subtract our compensation FFT from our input music FFT in order to

compensate for distortion. However, a more interesting DSP would be to perform an amount of compensation in real time. If you have speakers that are moving along a track into acoustically different areas that you wish to be able to compensation your music in real time to the area that you are in.