Advanced Digital Design Final Project: Digital Sound Effects Unit using FPGAs

Team: Christopher Scroasati, Isaac Ali

Submitted December 13, 2022

Project Overview and Objectives:

The objective of this project was to create an audio-effects unit, similar in functionality to a guitar pedal. The design took in an analog audio signal, digitally performed an effect on the audio, and output as an analog audio signal. We created four audio effects:

- Dynamic Range Compressor: reduce the output level of audio when above a fixed threshold based on a fixed ratio.
- 8-bit Converter: change sample magnitudes to match their 8-bit equivalent
- Stereo to Mono Converter: make the left and right audio signals match
- Bit-Crusher Distortion: modify every other sample to match in magnitude to the previous sample

Solution:

To create this design we needed to create three units. An analog to digital audio converter, a digital to analog audio converter, and the desired effects. The effects unit will consist of a free-state machine to determine the selected effect and the algorithms or functions which perform it. The A/D and D/A converters were modified based on code created by Altera to interface with their audio codec. The codec operated using 32-bit audio at a sample rate of 48 KHz.

System Diagram:

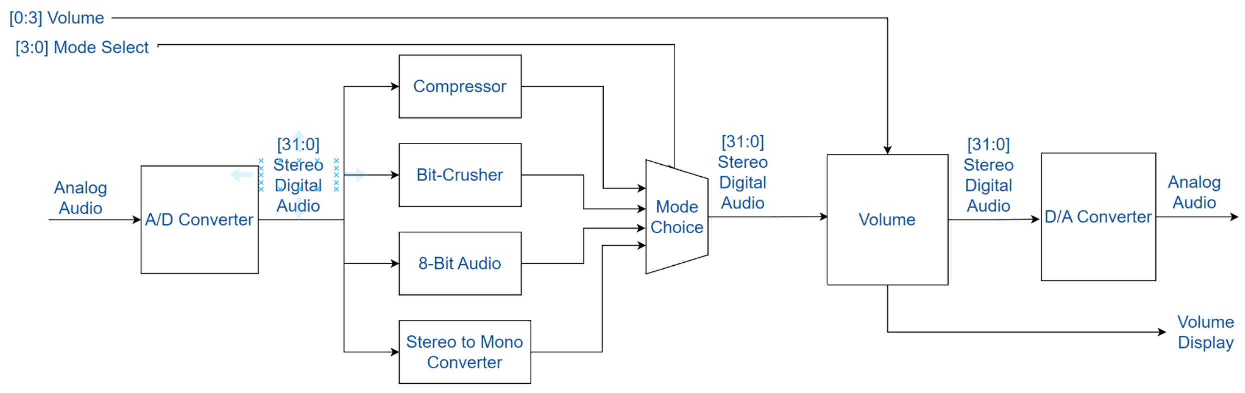


Figure 1: Block Diagram of Audio Effects System

Design Description:

The audio converter units used were provided by the altera IP catalog within the download for the given FPGA board. These units used an I2C communication protocol to interface with the audio codec’s data bus and interpret the given analog signals into digital signals with a bit depth of 32. These 32-bit samples were read at a frequency of 48 KHz. For our implementation purposes, this unit provided us a way to acquire the audio data which could be manipulated before being output.

To design the compressor we performed an averaging algorithm, similar in design to a synchronizer, when the magnitude of a sample exceeded a certain threshold. The synchronizer would chain values in stages which would update as new samples were taken. Every time this chain was updated it was divided by the ratio of audio compression desired.

```
always_comb begin
    if (inleft<threshold) begin//if left audio greater than threshold, compress audio
        outleft <= (inleft);
    end else begin
        outleft <= threshold+(lcomp-threshold);
    end
    if (inright<threshold) begin //if right audio greater than threshold, compress audio
        outright <= (inright);

    end else begin
        outright <= threshold+(rcomp-threshold);
    end
end
```

Figure 2: Compressor Algorithm - 'lcomp' and 'rcomp' are averaged values

To create the 8-bit audio effect we divided the input magnitude by a factor which converted the 32-bit value to it's 8-bit equivalent.

```
BIT8: begin //reduce resolution to 8-bit by dividing by 8-bit resultion
leftout<=leftin/(32'd256);
rightout<=rightin/(32'd256);
end
```

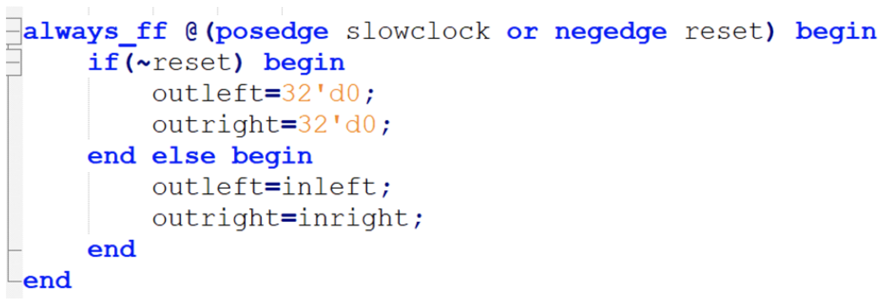
Figure 3: 8-bit Audio Algorithm

To design the stereo to mono converter we mapped the left audio signal to the right audio signal.

```
MONO: begin //stream mono audio by assigning left to right as well
leftout<=leftin;
rightout<=leftin;
end
```

Figure 4: Stereo to Mono Converter

To design the bit-crush audio effect we reduce the sample rate of the audio to make every 'x' amount of samples magnitudes be the same. The sample rate corresponds to the divided clock rate.



```
always_ff @(posedge slowclock or negedge reset) begin
    if(~reset) begin
        outleft=32'd0;
        outright=32'd0;
    end else begin
        outleft=inleft;
        outright=inright;
    end
end
```

Figure 5: Bit-Crush Algorithm - ‘slowclock’ has been clock divided

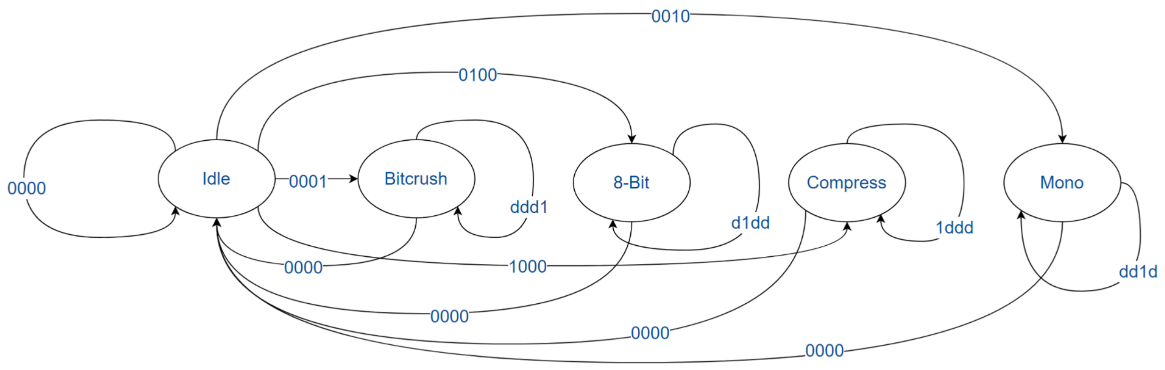


Figure 6: FSM for Effect Selection

The included design must be programmed onto an Altera DE2-115 FPGA. This is because of how the A/D and D/A converters must be created. To interface with an audio codec, you must know how the data bus processes audio to appropriately communicate and interpret the given information. This process also requires a specific clock rate to input and output audio. In signal processing terms this translates to the sample rate of the A/D converter for the input and the D/A converter for the output.

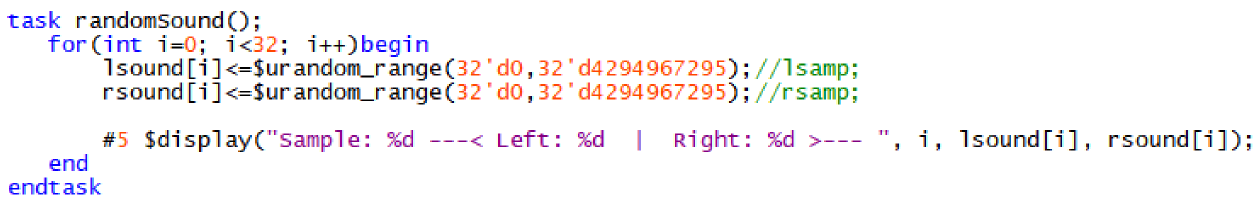
Testing Procedures:

Informal Testing

Initial, informal testing involved the implementation of Intel’s IP cores for Audio and Video. We plugged in a laptop to the line-in port on the FPGA, then connected a set of headphones to the line-out port on the board. The result was streamed audio from the laptop to the headphones with an unnoticeable amount of delay or latency.

Test Benching

With a general idea of how audio was meant to be passed through the system via the onboard ADC and DAC, we began planning a testing procedure. To start, we would need to randomize unsigned integers, then pass those numbers into an array that would represent a sequential and contiguous audio sample. The process is shown in Figure 1.

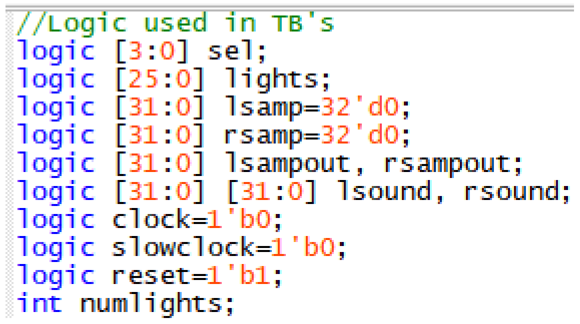


```
task randomSound();
   for(int i=0; i<32; i++)begin
       lsound[i]<=$urandom_range(32'd0,32'd4294967295);//lsamp;
       rsound[i]<=$urandom_range(32'd0,32'd4294967295);//rsamp;

       #5 $display("Sample: %d ---< Left: %d  |  Right: %d >--- ", i, lsound[i], rsound[i]);
   end
endtask
```

Figure 1: Random Audio Sample Generator

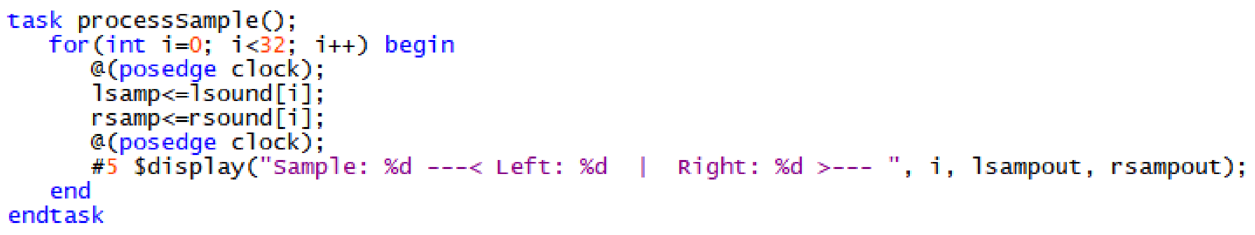
To do this we utilized “urandom_range” setting the min to 0 and max values to 2^32, since the audio stream being used by the system was 32 bits. To keep the values to a more realistic and reasonable level, the end result is divided by 300. Using a for loop and two packed 32-bit, 32 item arrays, we loaded the left and right samples independently into their respective containers. Displaying the values at each iteration of the for loop is very useful for comparing data sets. Another important consideration when designing this task was modularity. This task does not rely on module specific logic, and is instead placed in a separate SystemVerilog file with all the required logic declared in the System Verilog file, as shown in FIgure 2.



```
//Logic used in TB's
logic [3:0] sel;
logic [25:0] lights;
logic [31:0] lsamp=32'd0;
logic [31:0] rsamp=32'd0;
logic [31:0] lsampout, rsampout;
logic [31:0] [31:0] lsound, rsound;
logic clock=1'b0;
logic slowclock=1'b0;
logic reset=1'b1;
int numlights;
```

Figure 2: soundTask.sv logic declarations

Next, we needed a universal way to process the audio samples sequentially. By using a for loop and a clock edge detector, we can sequentially and synchronously process all of the data points in the previously created sound sample. This is shown in Figure 3.

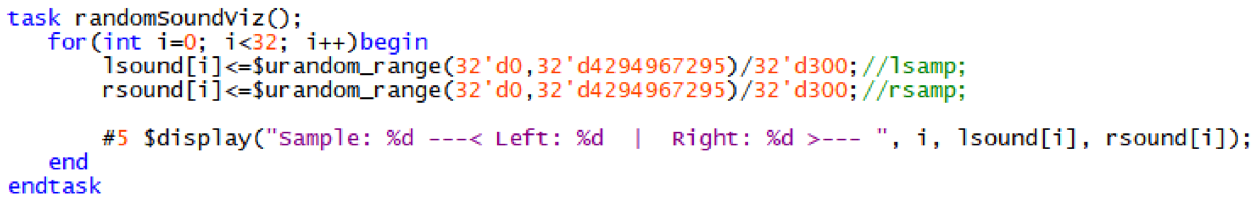


```
task processSample();
   for(int i=0; i<32; i++) begin
      @(posedge clock);
      lsamp<=lsound[i];
      rsamp<=rsound[i];
      @(posedge clock);
      #5 $display("Sample: %d ---< Left: %d  |  Right: %d >--- ", i, lsampout, rsampout);
   end
endtask
```

Figure 3: Audio Sample Processor

Two clock detectors are located inside of the for loop to account for potential latency in processing due to extra flip flops present in the effect modules. The display call will show the output from the system being tested, this task was also designed to be modular.

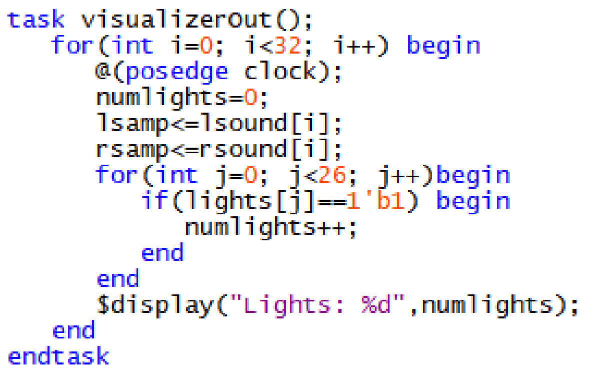
The first module to test was the visualizer. This proved to be a unique case where the function written in Figure 3 would not work for verification purposes. Instead, we copied the base framework and added support to output the visualizer's light states. Since we are using 26 lights, a nested for loop needed to be implemented in order to count the amount of lights that came on based off of the averaged value. The clock used to instantiate the module was simulated at a slower speed to make the change in light patterns visible when implemented on the board, also the values used when creating the sample were artificially reduced during testing to more clearly show the change in lights in a small pool of values. A separate randomizer was made for this, shown in Figure 4.



```
task randomSoundViz();
   for(int i=0; i<32; i++)begin
       lsound[i]<=$urandom_range(32'd0,32'd4294967295)/32'd300;//lsamp;
       rsound[i]<=$urandom_range(32'd0,32'd4294967295)/32'd300;//rsamp;

       #5 $display("Sample: %d ---< Left: %d  |  Right: %d >--- ", i, lsound[i], rsound[i]);
   end
endtask
```

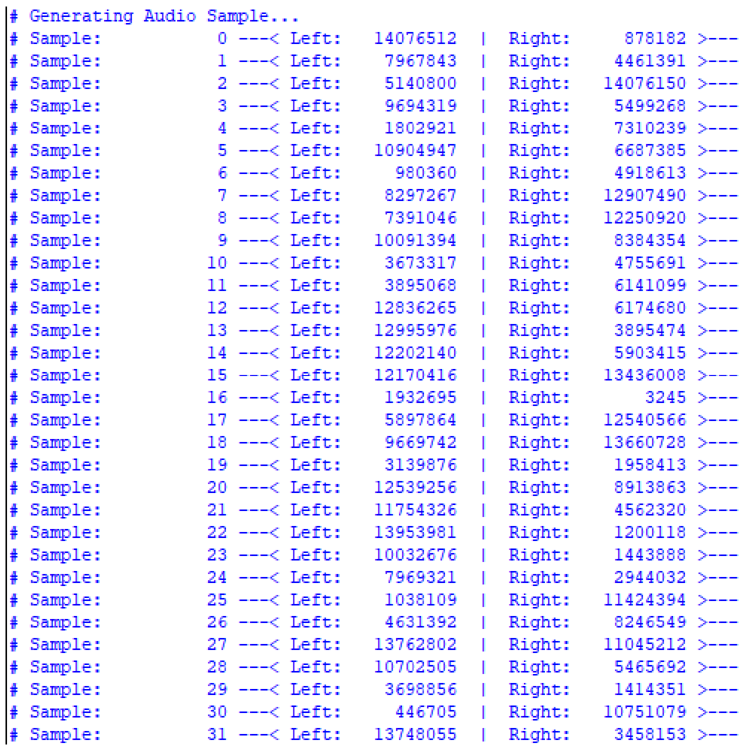
Figure 4: Random Audio Sample Generator For Visualizer



```
task visualizerOut();
   for(int i=0; i<32; i++) begin
      @(posedge clock);
      numlights=0;
      lsamp<=lsound[i];
      rsamp<=rsound[i];
      for(int j=0; j<26; j++)begin
         if(lights[j]==1'b1) begin
            numlights++;
         end
      end
      $display("Lights: %d",numlights);
   end
endtask
```

Figure 5: Visualizer Output Task

Figure 4 shows the code used to implement the task for the visualizer output verification. For each light that is on in the visualizer, "numlights" is incremented, representing the sequential lighting of the lights from the module. Figure 5 shows the sound sample created using urandom. Figure 6 shows the output of the testbench for the visualizer.

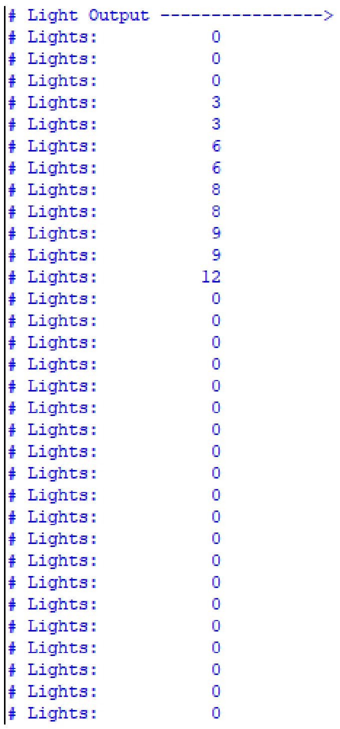


```
# Generating Audio Sample...
# Sample:          0 ---< Left:   14076512  |  Right:    878182 >---
# Sample:          1 ---< Left:    7967843  |  Right:   4461391 >---
# Sample:          2 ---< Left:    5140800  |  Right:  14076150 >---
# Sample:          3 ---< Left:    9694319  |  Right:   5499268 >---
# Sample:          4 ---< Left:    1802921  |  Right:   7310239 >---
# Sample:          5 ---< Left:   10904947  |  Right:   6687385 >---
# Sample:          6 ---< Left:     980360  |  Right:   4918613 >---
# Sample:          7 ---< Left:    8297267  |  Right:  12907490 >---
# Sample:          8 ---< Left:    7391046  |  Right:  12250920 >---
# Sample:          9 ---< Left:   10091394  |  Right:   8384354 >---
# Sample:         10 ---< Left:    3673317  |  Right:   4755691 >---
# Sample:         11 ---< Left:    3895068  |  Right:   6141099 >---
# Sample:         12 ---< Left:   12836265  |  Right:   6174680 >---
# Sample:         13 ---< Left:   12995976  |  Right:   3895474 >---
# Sample:         14 ---< Left:   12202140  |  Right:   5903415 >---
# Sample:         15 ---< Left:   12170416  |  Right:  13436008 >---
# Sample:         16 ---< Left:    1932695  |  Right:      3245 >---
# Sample:         17 ---< Left:    5897864  |  Right:  12540566 >---
# Sample:         18 ---< Left:    9669742  |  Right:  13660728 >---
# Sample:         19 ---< Left:    3139876  |  Right:   1958413 >---
# Sample:         20 ---< Left:   12539256  |  Right:   8913863 >---
# Sample:         21 ---< Left:   11754326  |  Right:   4562320 >---
# Sample:         22 ---< Left:   13953981  |  Right:   1200118 >---
# Sample:         23 ---< Left:   10032676  |  Right:   1443888 >---
# Sample:         24 ---< Left:    7969321  |  Right:   2944032 >---
# Sample:         25 ---< Left:    1038109  |  Right:  11424394 >---
# Sample:         26 ---< Left:    4631392  |  Right:   8246549 >---
# Sample:         27 ---< Left:   13762802  |  Right:  11045212 >---
# Sample:         28 ---< Left:   10702505  |  Right:   5465692 >---
# Sample:         29 ---< Left:    3698856  |  Right:   1414351 >---
# Sample:         30 ---< Left:     446705  |  Right:  10751079 >---
# Sample:         31 ---< Left:   13748055  |  Right:   3458153 >---
```

Figure 6: Audio Sample Transcript

<u>Results:</u>

The first testbench we ran for verification was the Visualizer test bench and confirmed correct operation of our audio visualizer.



```
# Light Output --------------->
# Lights:          0
# Lights:          0
# Lights:          0
# Lights:          3
# Lights:          3
# Lights:          6
# Lights:          6
# Lights:          8
# Lights:          8
# Lights:          9
# Lights:          9
# Lights:         12
# Lights:          0
# Lights:          0
# Lights:          0
# Lights:          0
# Lights:          0
# Lights:          0
# Lights:          0
# Lights:          0
# Lights:          0
# Lights:          0
# Lights:          0
# Lights:          0
# Lights:          0
# Lights:          0
# Lights:          0
# Lights:          0
# Lights:          0
# Lights:          0
# Lights:          0
# Lights:          0
```

Figure 1: Visualizer Test Bench Lights Output Transcript

As we can see in Figure 1, the first few initial outputs are 0 even though the samples shown in Figure 5 are non-zero numbers. This is due to the averaging function not having enough samples to process the average magnitude of the previous samples. The reset is also activated about half-way through on this testbench, leading to the absence of values in the output. This is expected behavior. Figure 2 shows the waves generated from this test bench.

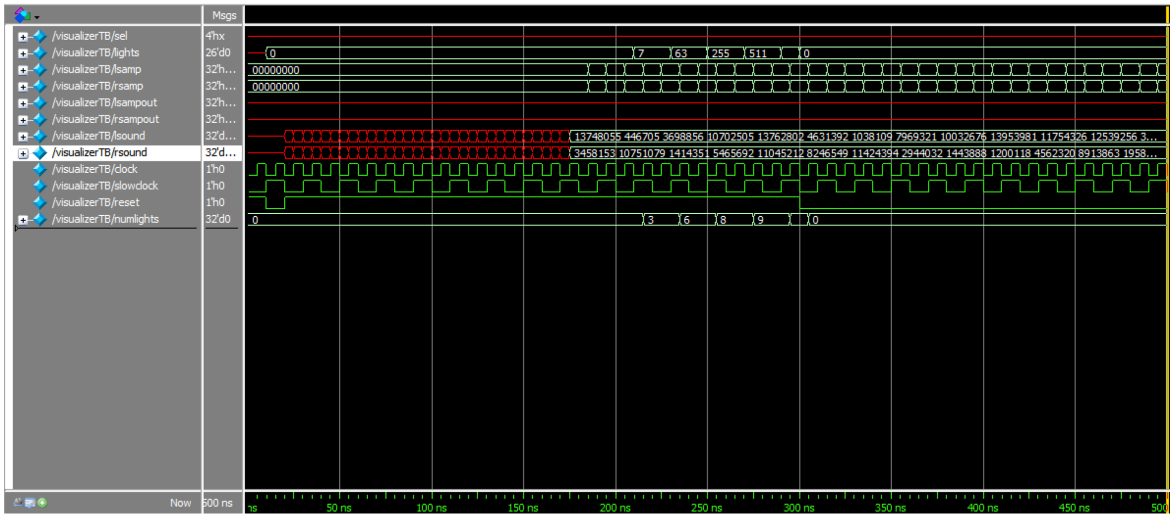
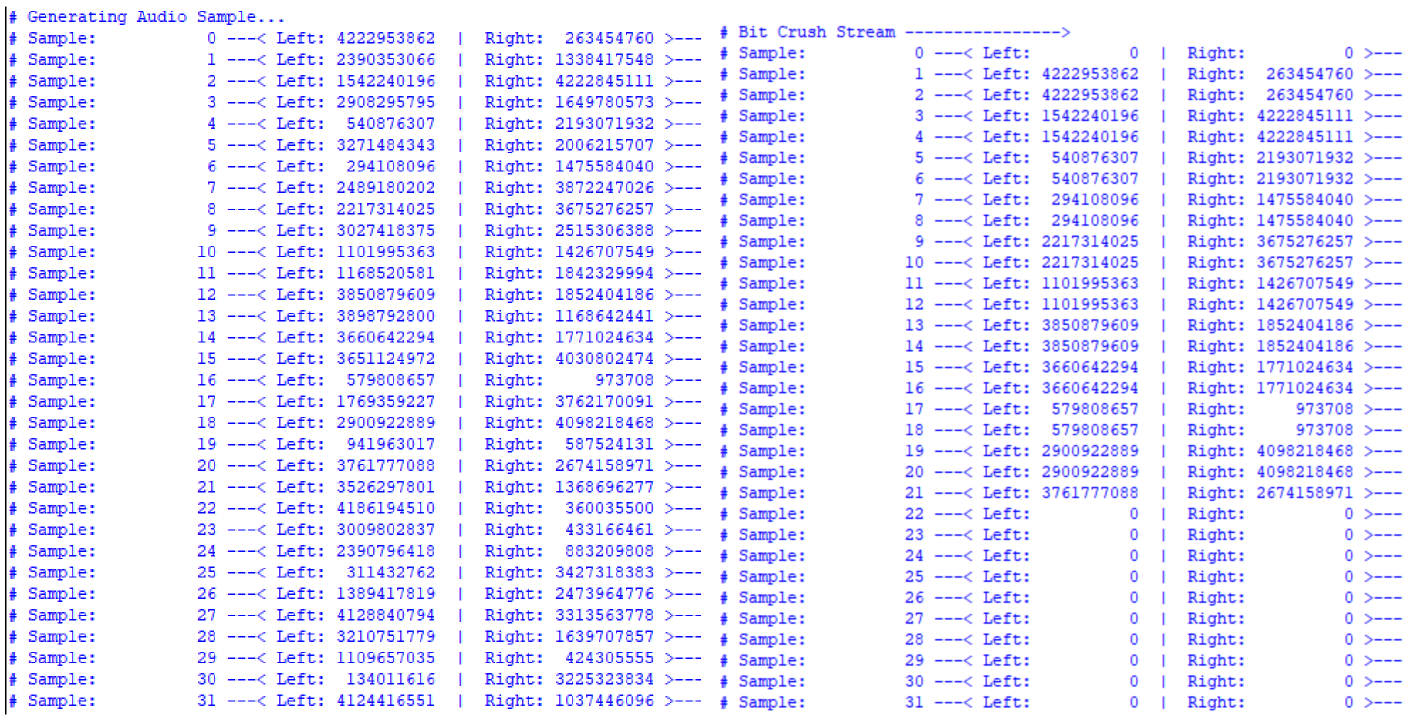


Figure 2: Visualizer Test Bench Waves

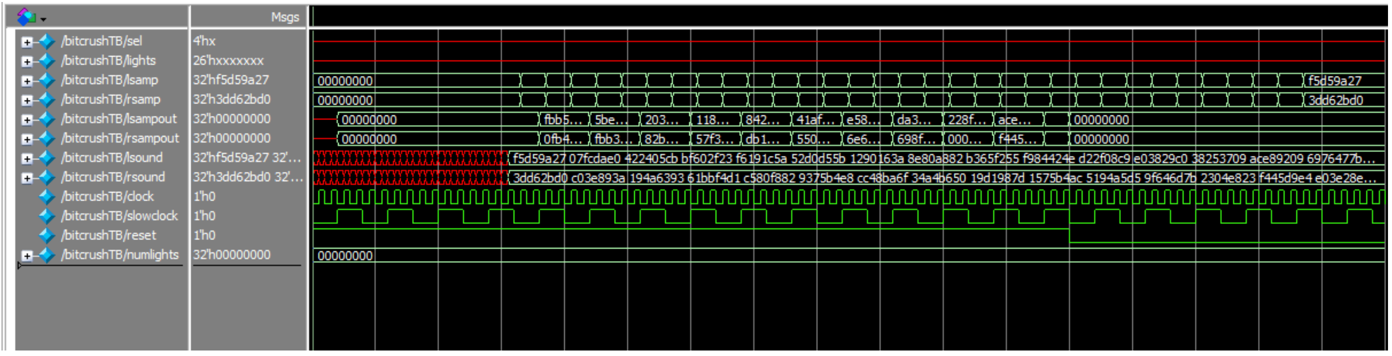
The next step was to test the audio effect modules. There are two that are not embedded into the FSM directly. Luckily, due to the modularity of our test bench design, we can test both in a near identical fashion. The bitcrush module functions using a separate, slowclock variable used to simulate the slowed clock in use by the system to make the effect possible. Figure 3 shows the output of the bitcrush testbench.



```
# Generating Audio Sample...
# Sample:           0 ---< Left: 4222953862  |  Right:  263454760 >---
# Sample:           1 ---< Left: 2390353066  |  Right: 1338417548 >---
# Sample:           2 ---< Left: 1542240196  |  Right: 4222845111 >---
# Sample:           3 ---< Left: 2908295795  |  Right: 1649780573 >---
# Sample:           4 ---< Left:  540876307  |  Right: 2193071932 >---
# Sample:           5 ---< Left: 3271484343  |  Right: 2006215707 >---
# Sample:           6 ---< Left:  294108096  |  Right: 1475584040 >---
# Sample:           7 ---< Left: 2489180202  |  Right: 3872247026 >---
# Sample:           8 ---< Left: 2217314025  |  Right: 3675276257 >---
# Sample:           9 ---< Left: 3027418375  |  Right: 2515306388 >---
# Sample:          10 ---< Left: 1101995363  |  Right: 1426707549 >---
# Sample:          11 ---< Left: 1168520581  |  Right: 1842329994 >---
# Sample:          12 ---< Left: 3850879609  |  Right: 1852404186 >---
# Sample:          13 ---< Left: 3898792800  |  Right: 1168642441 >---
# Sample:          14 ---< Left: 3660642294  |  Right: 1771024634 >---
# Sample:          15 ---< Left: 3651124972  |  Right: 4030802474 >---
# Sample:          16 ---< Left:  579808657  |  Right:     973708 >---
# Sample:          17 ---< Left: 1769359227  |  Right: 3762170091 >---
# Sample:          18 ---< Left: 2900922889  |  Right: 4098218468 >---
# Sample:          19 ---< Left:  941963017  |  Right:  587524131 >---
# Sample:          20 ---< Left: 3761777088  |  Right: 2674158971 >---
# Sample:          21 ---< Left: 3526297801  |  Right: 1368696277 >---
# Sample:          22 ---< Left: 4186194510  |  Right:  360035500 >---
# Sample:          23 ---< Left: 3009802837  |  Right:  433166461 >---
# Sample:          24 ---< Left: 2390796418  |  Right:  883209808 >---
# Sample:          25 ---< Left:  311432762  |  Right: 3427318383 >---
# Sample:          26 ---< Left: 1389417819  |  Right: 2473964776 >---
# Sample:          27 ---< Left: 4128840794  |  Right: 3313563778 >---
# Sample:          28 ---< Left: 3210751779  |  Right: 1639707857 >---
# Sample:          29 ---< Left: 1109657035  |  Right:  424305555 >---
# Sample:          30 ---< Left:  134011616  |  Right: 3225323834 >---
# Sample:          31 ---< Left: 4124416551  |  Right: 1037446096 >---
```

```
# Bit Crush Stream ---------------->
# Sample:           0 ---< Left:          0  |  Right:          0 >---
# Sample:           1 ---< Left: 4222953862  |  Right:  263454760 >---
# Sample:           2 ---< Left: 4222953862  |  Right:  263454760 >---
# Sample:           3 ---< Left: 1542240196  |  Right: 4222845111 >---
# Sample:           4 ---< Left: 1542240196  |  Right: 4222845111 >---
# Sample:           5 ---< Left:  540876307  |  Right: 2193071932 >---
# Sample:           6 ---< Left:  540876307  |  Right: 2193071932 >---
# Sample:           7 ---< Left:  294108096  |  Right: 1475584040 >---
# Sample:           8 ---< Left:  294108096  |  Right: 1475584040 >---
# Sample:           9 ---< Left: 2217314025  |  Right: 3675276257 >---
# Sample:          10 ---< Left: 2217314025  |  Right: 3675276257 >---
# Sample:          11 ---< Left: 1101995363  |  Right: 1426707549 >---
# Sample:          12 ---< Left: 1101995363  |  Right: 1426707549 >---
# Sample:          13 ---< Left: 3850879609  |  Right: 1852404186 >---
# Sample:          14 ---< Left: 3850879609  |  Right: 1852404186 >---
# Sample:          15 ---< Left: 3660642294  |  Right: 1771024634 >---
# Sample:          16 ---< Left: 3660642294  |  Right: 1771024634 >---
# Sample:          17 ---< Left:  579808657  |  Right:     973708 >---
# Sample:          18 ---< Left:  579808657  |  Right:     973708 >---
# Sample:          19 ---< Left: 2900922889  |  Right: 4098218468 >---
# Sample:          20 ---< Left: 2900922889  |  Right: 4098218468 >---
# Sample:          21 ---< Left: 3761777088  |  Right: 2674158971 >---
# Sample:          22 ---< Left:          0  |  Right:          0 >---
# Sample:          23 ---< Left:          0  |  Right:          0 >---
# Sample:          24 ---< Left:          0  |  Right:          0 >---
# Sample:          25 ---< Left:          0  |  Right:          0 >---
# Sample:          26 ---< Left:          0  |  Right:          0 >---
# Sample:          27 ---< Left:          0  |  Right:          0 >---
# Sample:          28 ---< Left:          0  |  Right:          0 >---
# Sample:          29 ---< Left:          0  |  Right:          0 >---
# Sample:          30 ---< Left:          0  |  Right:          0 >---
# Sample:          31 ---< Left:          0  |  Right:          0 >---
```

Figure 3: Bitcrusher Test Bench Transcript

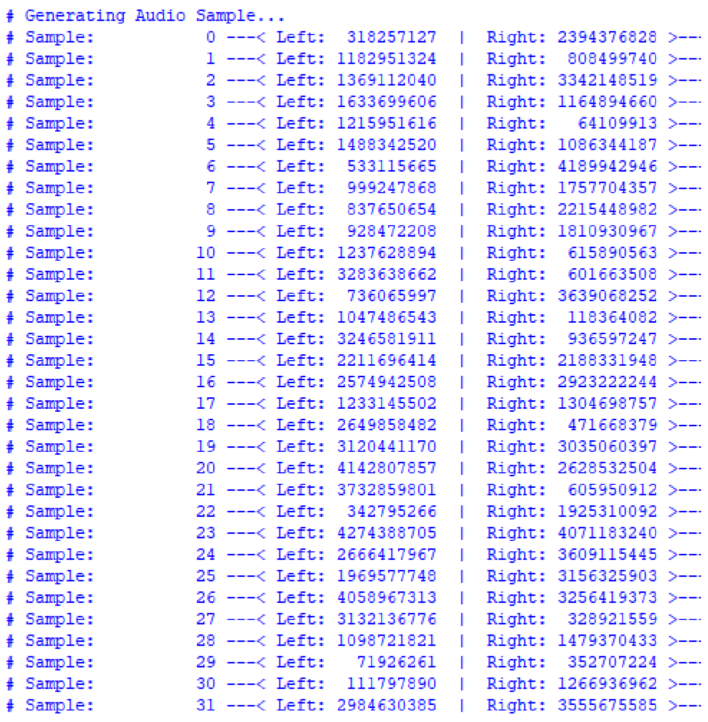


```
                          Msgs
/bitcrushTB/sel           4'hx
/bitcrushTB/lights        26'hxxxxxxx
/bitcrushTB/lsamp         32'hf5d59a27
/bitcrushTB/rsamp         32'h3dd62bd0
/bitcrushTB/lsampout      32'h00000000
/bitcrushTB/rsampout      32'h00000000
/bitcrushTB/lsound        32'hf5d59a27 32'...
/bitcrushTB/rsound        32'h3dd62bd0 32'...
/bitcrushTB/clock         1'h0
/bitcrushTB/slowclock     1'h0
/bitcrushTB/reset         1'h0
/bitcrushTB/numlights     32'h00000000
```

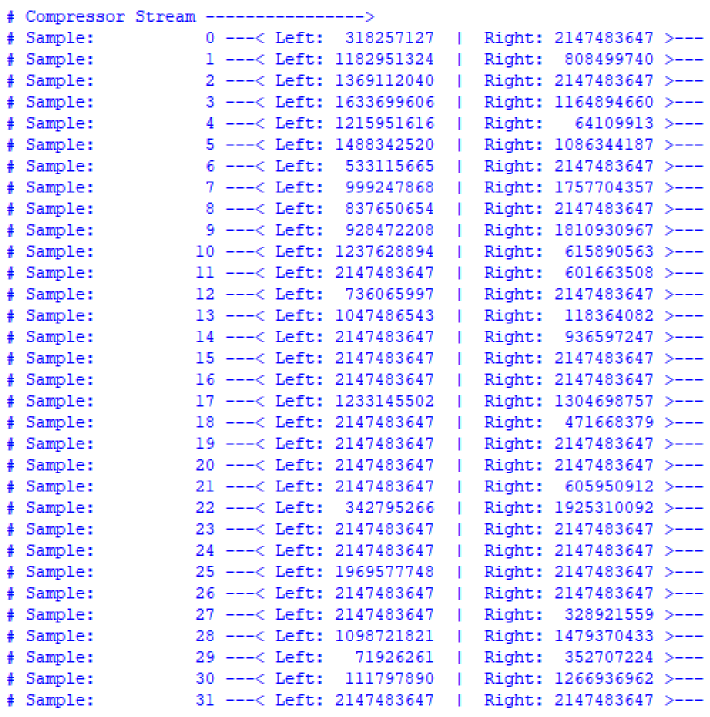
00000000
f5d59a27
3dd62bd0
fbb5... 5be... 203... 118... 842... 41af... e58... da3... 228f... ace...
0fb4... fbb3... 82b... 57f3... db1... 550... 6e6... 698f... 000... f445...
f5d59a27 07fcdae0 422405cb bf602f23 f6191c5a 52d0d55b 1290163a 8e80a882 b365f255 f984424e d22f08c9 e03829c0 38253709 ace89209 6976477b...
3dd62bd0 c03e893a 194a6393 61bbf4d1 c580f882 9375b4e8 cc48ba6f 34a4b650 19d1987d 1575b4ac 5194a5d5 9f646d7b 2304e823 f445d9e4 e03e28e...

Figure 4: Bitcrusher Test Bench Waves

This test bench works by continuously forcing two clock signals, one for the normal system clock, and one for the slowed clock used to create the bitcrushed audio. The effect of bitcrushing is evidenced by the repeated sample values, this is due to the slower clock driving the audio output. The reset is triggered about halfway through the test bench. Figure 4 shows when the reset was triggered. The transcript window for the compressor test bench is shown in figure 5. Figure 6 contains the waves for the compressor test bench. One oddity we ran into while working on the test bench for the compressor is that the compressor would only work on the testbench when a variable was removed from the equation, but then would not work on the board.

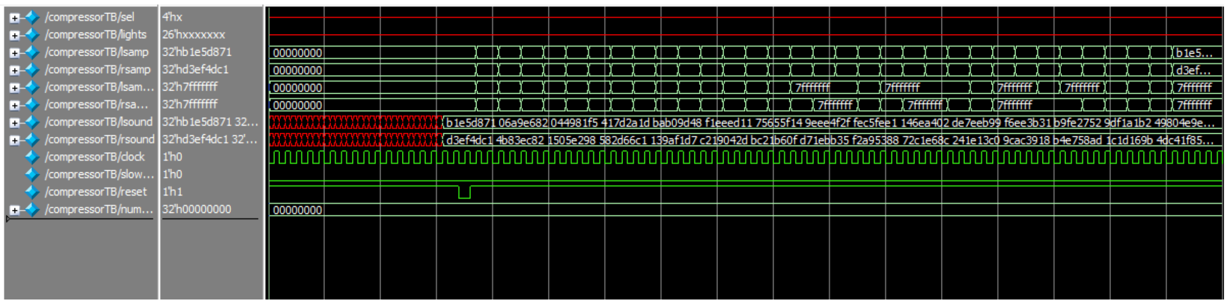


```
# Generating Audio Sample...
# Sample:            0 ---< Left:  318257127  |  Right: 2394376828 >---
# Sample:            1 ---< Left: 1182951324  |  Right:  808499740 >---
# Sample:            2 ---< Left: 1369112040  |  Right: 3342148519 >---
# Sample:            3 ---< Left: 1633699606  |  Right: 1164894660 >---
# Sample:            4 ---< Left: 1215951616  |  Right:   64109913 >---
# Sample:            5 ---< Left: 1488342520  |  Right: 1086344187 >---
# Sample:            6 ---< Left:  533115665  |  Right: 4189942946 >---
# Sample:            7 ---< Left:  999247868  |  Right: 1757704357 >---
# Sample:            8 ---< Left:  837650654  |  Right: 2215448982 >---
# Sample:            9 ---< Left:  928472208  |  Right: 1810930967 >---
# Sample:           10 ---< Left: 1237628894  |  Right:  615890563 >---
# Sample:           11 ---< Left: 3283638662  |  Right:  601663508 >---
# Sample:           12 ---< Left:  736065997  |  Right: 3639068252 >---
# Sample:           13 ---< Left: 1047486543  |  Right:  118364082 >---
# Sample:           14 ---< Left: 3246581911  |  Right:  936597247 >---
# Sample:           15 ---< Left: 2211696414  |  Right: 2188331948 >---
# Sample:           16 ---< Left: 2574942508  |  Right: 2923222244 >---
# Sample:           17 ---< Left: 1233145502  |  Right: 1304698757 >---
# Sample:           18 ---< Left: 2649858482  |  Right:  471668379 >---
# Sample:           19 ---< Left: 3120441170  |  Right: 3035060397 >---
# Sample:           20 ---< Left: 4142807857  |  Right: 2628532504 >---
# Sample:           21 ---< Left: 3732859801  |  Right:  605950912 >---
# Sample:           22 ---< Left:  342795266  |  Right: 1925310092 >---
# Sample:           23 ---< Left: 4274388705  |  Right: 4071183240 >---
# Sample:           24 ---< Left: 2666417967  |  Right: 3609115445 >---
# Sample:           25 ---< Left: 1969577748  |  Right: 3156325903 >---
# Sample:           26 ---< Left: 4058967313  |  Right: 3256419373 >---
# Sample:           27 ---< Left: 3132136776  |  Right:  328921559 >---
# Sample:           28 ---< Left: 1098721821  |  Right: 1479370433 >---
# Sample:           29 ---< Left:   71926261  |  Right:  352707224 >---
# Sample:           30 ---< Left:  111797890  |  Right: 1266936962 >---
# Sample:           31 ---< Left: 2984630385  |  Right: 3555675585 >---
```



```
# Compressor Stream ---------------->
# Sample:            0 ---< Left:  318257127  |  Right: 2147483647 >---
# Sample:            1 ---< Left: 1182951324  |  Right:  808499740 >---
# Sample:            2 ---< Left: 1369112040  |  Right: 2147483647 >---
# Sample:            3 ---< Left: 1633699606  |  Right: 1164894660 >---
# Sample:            4 ---< Left: 1215951616  |  Right:   64109913 >---
# Sample:            5 ---< Left: 1488342520  |  Right: 1086344187 >---
# Sample:            6 ---< Left:  533115665  |  Right: 2147483647 >---
# Sample:            7 ---< Left:  999247868  |  Right: 1757704357 >---
# Sample:            8 ---< Left:  837650654  |  Right: 2147483647 >---
# Sample:            9 ---< Left:  928472208  |  Right: 1810930967 >---
# Sample:           10 ---< Left: 1237628894  |  Right:  615890563 >---
# Sample:           11 ---< Left: 2147483647  |  Right:  601663508 >---
# Sample:           12 ---< Left:  736065997  |  Right: 2147483647 >---
# Sample:           13 ---< Left: 1047486543  |  Right:  118364082 >---
# Sample:           14 ---< Left: 2147483647  |  Right:  936597247 >---
# Sample:           15 ---< Left: 2147483647  |  Right: 2147483647 >---
# Sample:           16 ---< Left: 2147483647  |  Right: 2147483647 >---
# Sample:           17 ---< Left: 1233145502  |  Right: 1304698757 >---
# Sample:           18 ---< Left: 2147483647  |  Right:  471668379 >---
# Sample:           19 ---< Left: 2147483647  |  Right: 2147483647 >---
# Sample:           20 ---< Left: 2147483647  |  Right: 2147483647 >---
# Sample:           21 ---< Left: 2147483647  |  Right:  605950912 >---
# Sample:           22 ---< Left:  342795266  |  Right: 1925310092 >---
# Sample:           23 ---< Left: 2147483647  |  Right: 2147483647 >---
# Sample:           24 ---< Left: 2147483647  |  Right: 2147483647 >---
# Sample:           25 ---< Left: 1969577748  |  Right: 2147483647 >---
# Sample:           26 ---< Left: 2147483647  |  Right: 2147483647 >---
# Sample:           27 ---< Left: 2147483647  |  Right:  328921559 >---
# Sample:           28 ---< Left: 1098721821  |  Right: 1479370433 >---
# Sample:           29 ---< Left:   71926261  |  Right:  352707224 >---
# Sample:           30 ---< Left:  111797890  |  Right: 1266936962 >---
# Sample:           31 ---< Left: 2147483647  |  Right: 2147483647 >---
```

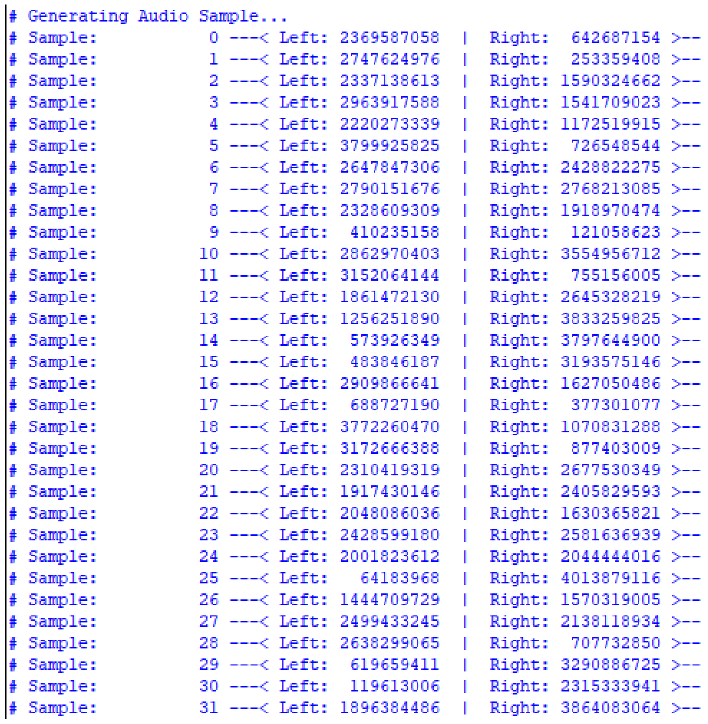
Figure 5: Compressor Test Bench Transcript



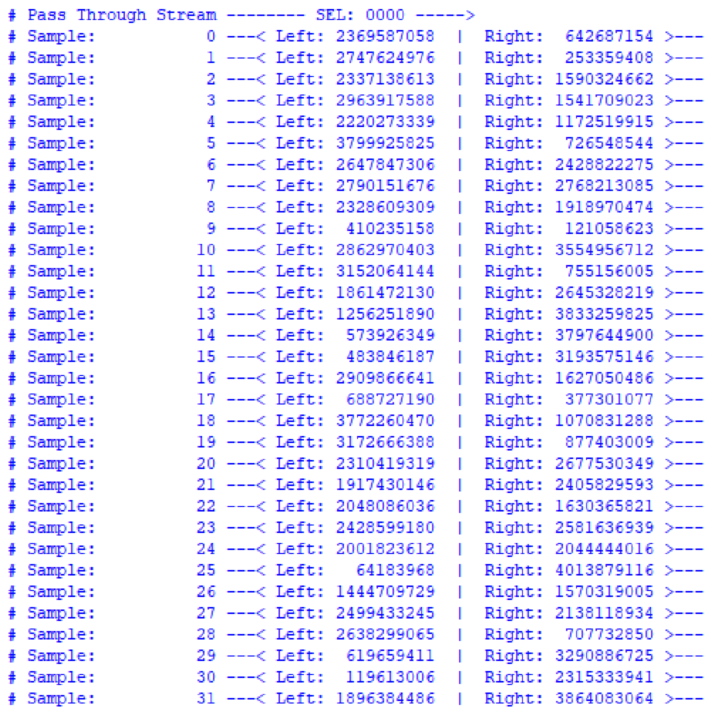
```
/compressorTB/sel       4'hx
/compressorTB/lights    26'hxxxxxxxx
/compressorTB/lsamp     32'hb1e5d871
/compressorTB/rsamp     32'hd3ef4dc1
/compressorTB/lsam...   32'h7fffffff
/compressorTB/rsa...    32'h7fffffff
/compressorTB/lsound    32'hb1e5d871 32...
/compressorTB/rsound    32'hd3ef4dc1 32'...
/compressorTB/clock     1'h0
/compressorTB/slow...   1'h0
/compressorTB/reset     1'h1
/compressorTB/num...    32'h00000000
```

Figure 6: Compressor Test Bench Waves

The last testbench we wrote was for the audio effects FSM. This testbench would need to apply the individual audio effects using the selection variable. First we would generate an audio sample, then we would apply each effect. Figures 7 and 8 show the audio sample generation, the pass through function output, mono output, and the 8-bit resolution output.

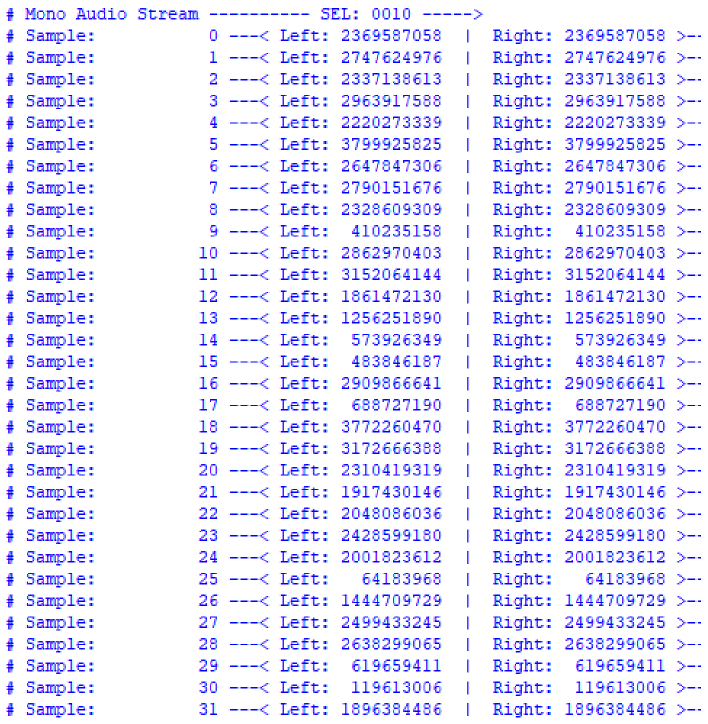


```
# Generating Audio Sample...
# Sample:            0 ---< Left: 2369587058  |  Right:  642687154 >--
# Sample:            1 ---< Left: 2747624976  |  Right:  253359408 >--
# Sample:            2 ---< Left: 2337138613  |  Right: 1590324662 >--
# Sample:            3 ---< Left: 2963917588  |  Right: 1541709023 >--
# Sample:            4 ---< Left: 2220273339  |  Right: 1172519915 >--
# Sample:            5 ---< Left: 3799925825  |  Right:  726548544 >--
# Sample:            6 ---< Left: 2647847306  |  Right: 2428822275 >--
# Sample:            7 ---< Left: 2790151676  |  Right: 2768213085 >--
# Sample:            8 ---< Left: 2328609309  |  Right: 1918970474 >--
# Sample:            9 ---< Left:  410235158  |  Right:  121058623 >--
# Sample:           10 ---< Left: 2862970403  |  Right: 3554956712 >--
# Sample:           11 ---< Left: 3152064144  |  Right:  755156005 >--
# Sample:           12 ---< Left: 1861472130  |  Right: 2645328219 >--
# Sample:           13 ---< Left: 1256251890  |  Right: 3833259825 >--
# Sample:           14 ---< Left:  573926349  |  Right: 3797644900 >--
# Sample:           15 ---< Left:  483846187  |  Right: 3193575146 >--
# Sample:           16 ---< Left: 2909866641  |  Right: 1627050486 >--
# Sample:           17 ---< Left:  688727190  |  Right:  377301077 >--
# Sample:           18 ---< Left: 3772260470  |  Right: 1070831288 >--
# Sample:           19 ---< Left: 3172666388  |  Right:  877403009 >--
# Sample:           20 ---< Left: 2310419319  |  Right: 2677530349 >--
# Sample:           21 ---< Left: 1917430146  |  Right: 2405829593 >--
# Sample:           22 ---< Left: 2048086036  |  Right: 1630365821 >--
# Sample:           23 ---< Left: 2428599180  |  Right: 2581636939 >--
# Sample:           24 ---< Left: 2001823612  |  Right: 2044444016 >--
# Sample:           25 ---< Left:   64183968  |  Right: 4013879116 >--
# Sample:           26 ---< Left: 1444709729  |  Right: 1570319005 >--
# Sample:           27 ---< Left: 2499433245  |  Right: 2138118934 >--
# Sample:           28 ---< Left: 2638299065  |  Right:  707732850 >--
# Sample:           29 ---< Left:  619659411  |  Right: 3290886725 >--
# Sample:           30 ---< Left:  119613006  |  Right: 2315333941 >--
# Sample:           31 ---< Left: 1896384486  |  Right: 3864083064 >--
```

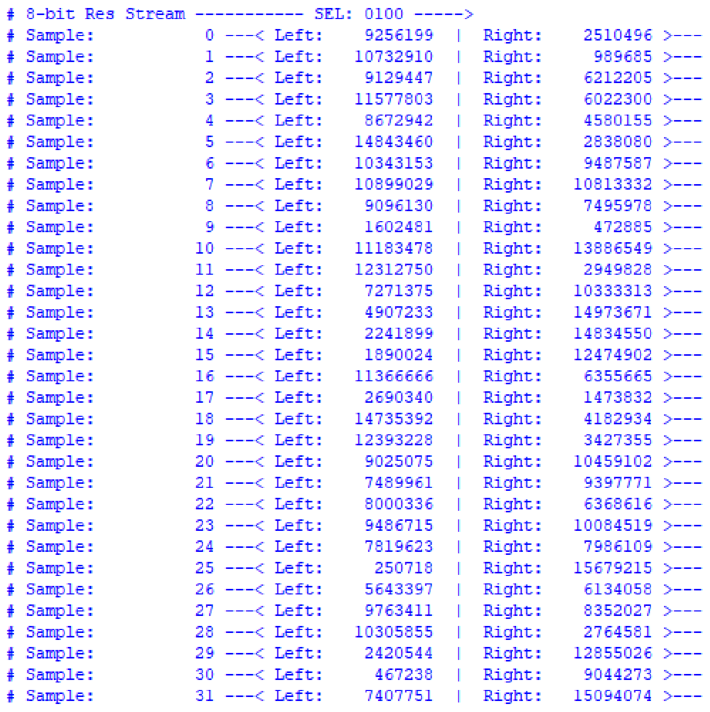


```
# Pass Through Stream -------- SEL: 0000 ----->
# Sample:            0 ---< Left: 2369587058  |  Right:  642687154 >---
# Sample:            1 ---< Left: 2747624976  |  Right:  253359408 >---
# Sample:            2 ---< Left: 2337138613  |  Right: 1590324662 >---
# Sample:            3 ---< Left: 2963917588  |  Right: 1541709023 >---
# Sample:            4 ---< Left: 2220273339  |  Right: 1172519915 >---
# Sample:            5 ---< Left: 3799925825  |  Right:  726548544 >---
# Sample:            6 ---< Left: 2647847306  |  Right: 2428822275 >---
# Sample:            7 ---< Left: 2790151676  |  Right: 2768213085 >---
# Sample:            8 ---< Left: 2328609309  |  Right: 1918970474 >---
# Sample:            9 ---< Left:  410235158  |  Right:  121058623 >---
# Sample:           10 ---< Left: 2862970403  |  Right: 3554956712 >---
# Sample:           11 ---< Left: 3152064144  |  Right:  755156005 >---
# Sample:           12 ---< Left: 1861472130  |  Right: 2645328219 >---
# Sample:           13 ---< Left: 1256251890  |  Right: 3833259825 >---
# Sample:           14 ---< Left:  573926349  |  Right: 3797644900 >---
# Sample:           15 ---< Left:  483846187  |  Right: 3193575146 >---
# Sample:           16 ---< Left: 2909866641  |  Right: 1627050486 >---
# Sample:           17 ---< Left:  688727190  |  Right:  377301077 >---
# Sample:           18 ---< Left: 3772260470  |  Right: 1070831288 >---
# Sample:           19 ---< Left: 3172666388  |  Right:  877403009 >---
# Sample:           20 ---< Left: 2310419319  |  Right: 2677530349 >---
# Sample:           21 ---< Left: 1917430146  |  Right: 2405829593 >---
# Sample:           22 ---< Left: 2048086036  |  Right: 1630365821 >---
# Sample:           23 ---< Left: 2428599180  |  Right: 2581636939 >---
# Sample:           24 ---< Left: 2001823612  |  Right: 2044444016 >---
# Sample:           25 ---< Left:   64183968  |  Right: 4013879116 >---
# Sample:           26 ---< Left: 1444709729  |  Right: 1570319005 >---
# Sample:           27 ---< Left: 2499433245  |  Right: 2138118934 >---
# Sample:           28 ---< Left: 2638299065  |  Right:  707732850 >---
# Sample:           29 ---< Left:  619659411  |  Right: 3290886725 >---
# Sample:           30 ---< Left:  119613006  |  Right: 2315333941 >---
# Sample:           31 ---< Left: 1896384486  |  Right: 3864083064 >---
```

Figure 7: Audio Sample Generation and Pass through



```
# Mono Audio Stream ---------- SEL: 0010 ----->
# Sample:            0 ---< Left: 2369587058  |  Right: 2369587058 >--
# Sample:            1 ---< Left: 2747624976  |  Right: 2747624976 >--
# Sample:            2 ---< Left: 2337138613  |  Right: 2337138613 >--
# Sample:            3 ---< Left: 2963917588  |  Right: 2963917588 >--
# Sample:            4 ---< Left: 2220273339  |  Right: 2220273339 >--
# Sample:            5 ---< Left: 3799925825  |  Right: 3799925825 >--
# Sample:            6 ---< Left: 2647847306  |  Right: 2647847306 >--
# Sample:            7 ---< Left: 2790151676  |  Right: 2790151676 >--
# Sample:            8 ---< Left: 2328609309  |  Right: 2328609309 >--
# Sample:            9 ---< Left:  410235158  |  Right:  410235158 >--
# Sample:           10 ---< Left: 2862970403  |  Right: 2862970403 >--
# Sample:           11 ---< Left: 3152064144  |  Right: 3152064144 >--
# Sample:           12 ---< Left: 1861472130  |  Right: 1861472130 >--
# Sample:           13 ---< Left: 1256251890  |  Right: 1256251890 >--
# Sample:           14 ---< Left:  573926349  |  Right:  573926349 >--
# Sample:           15 ---< Left:  483846187  |  Right:  483846187 >--
# Sample:           16 ---< Left: 2909866641  |  Right: 2909866641 >--
# Sample:           17 ---< Left:  688727190  |  Right:  688727190 >--
# Sample:           18 ---< Left: 3772260470  |  Right: 3772260470 >--
# Sample:           19 ---< Left: 3172666388  |  Right: 3172666388 >--
# Sample:           20 ---< Left: 2310419319  |  Right: 2310419319 >--
# Sample:           21 ---< Left: 1917430146  |  Right: 1917430146 >--
# Sample:           22 ---< Left: 2048086036  |  Right: 2048086036 >--
# Sample:           23 ---< Left: 2428599180  |  Right: 2428599180 >--
# Sample:           24 ---< Left: 2001823612  |  Right: 2001823612 >--
# Sample:           25 ---< Left:   64183968  |  Right:   64183968 >--
# Sample:           26 ---< Left: 1444709729  |  Right: 1444709729 >--
# Sample:           27 ---< Left: 2499433245  |  Right: 2499433245 >--
# Sample:           28 ---< Left: 2638299065  |  Right: 2638299065 >--
# Sample:           29 ---< Left:  619659411  |  Right:  619659411 >--
# Sample:           30 ---< Left:  119613006  |  Right:  119613006 >--
# Sample:           31 ---< Left: 1896384486  |  Right: 1896384486 >--
```



```
# 8-bit Res Stream ----------- SEL: 0100 ----->
# Sample:            0 ---< Left:    9256199  |  Right:    2510496 >---
# Sample:            1 ---< Left:   10732910  |  Right:     989685 >---
# Sample:            2 ---< Left:    9129447  |  Right:    6212205 >---
# Sample:            3 ---< Left:   11577803  |  Right:    6022300 >---
# Sample:            4 ---< Left:    8672942  |  Right:    4580155 >---
# Sample:            5 ---< Left:   14843460  |  Right:    2838080 >---
# Sample:            6 ---< Left:   10343153  |  Right:    9487587 >---
# Sample:            7 ---< Left:   10899029  |  Right:   10813332 >---
# Sample:            8 ---< Left:    9096130  |  Right:    7495978 >---
# Sample:            9 ---< Left:    1602481  |  Right:     472885 >---
# Sample:           10 ---< Left:   11183478  |  Right:   13886549 >---
# Sample:           11 ---< Left:   12312750  |  Right:    2949828 >---
# Sample:           12 ---< Left:    7271375  |  Right:   10333313 >---
# Sample:           13 ---< Left:    4907233  |  Right:   14973671 >---
# Sample:           14 ---< Left:    2241899  |  Right:   14834550 >---
# Sample:           15 ---< Left:    1890024  |  Right:   12474902 >---
# Sample:           16 ---< Left:   11366666  |  Right:    6355665 >---
# Sample:           17 ---< Left:    2690340  |  Right:    1473832 >---
# Sample:           18 ---< Left:   14735392  |  Right:    4182934 >---
# Sample:           19 ---< Left:   12393228  |  Right:    3427355 >---
# Sample:           20 ---< Left:    9025075  |  Right:   10459102 >---
# Sample:           21 ---< Left:    7489961  |  Right:    9397771 >---
# Sample:           22 ---< Left:    8000336  |  Right:    6368616 >---
# Sample:           23 ---< Left:    9486715  |  Right:   10084519 >---
# Sample:           24 ---< Left:    7819623  |  Right:    7986109 >---
# Sample:           25 ---< Left:     250718  |  Right:   15679215 >---
# Sample:           26 ---< Left:    5643397  |  Right:    6134058 >---
# Sample:           27 ---< Left:    9763411  |  Right:    8352027 >---
# Sample:           28 ---< Left:   10305855  |  Right:    2764581 >---
# Sample:           29 ---< Left:    2420544  |  Right:   12855026 >---
# Sample:           30 ---< Left:     467238  |  Right:    9044273 >---
# Sample:           31 ---< Left:    7407751  |  Right:   15094074 >---
```

Figure 8: Mono Audio effect and 8-bit Resolution effect

Our final design met all of our design specifications. We were able to perform effects on audio passed through the FPGA as expected. The most challenging part of this project for us was gaining our footing in a new area of knowledge. We had to learn how an audio codec works, and what that meant from a design standpoint. We also had to learn how to process real-time audio data in a way that modeled effects made popular by analog circuitry.

Analysis:

Our design worked fairly well, given that it was an introduction to audio signal processing using an FPGA. Functionally there were no unintended features, besides any which were a result of hardware limitations. The area needing most improvement would be audio fidelity and quality of effect algorithms.

When utilizing the FPGAs' codec you are required to utilize system variables. These variables are passed through into the audio conversion modules. Because of this, we were not able to perform timing analysis on our design. We utilized bidirectional innouts within our implementation, and we only know how to integrate inputs and outputs onto SDC files. The compilation fails using an SDC, citing the bidirectional inputs as constant values.

Audio quality within our design left much to be desired. There are two factors holding back the quality of sound at the moment, and both are having a noticeable impact on the results. The first is a clock rate disparity or skew which is causing audible bit distortion in all modes, including when audio is only being passed through our design. Having all clocks in sync with each other could help solve this issue.

The second is the usability of data provided by the codec. We assumed 32 bit unsigned logics as data for each sample, because this was the concept that brought us the most success. But unfortunately this information was not documented anywhere to be confirmed. This is important because it determines algorithms used for some of the effects. Knowing the correct format of the data could solve some of the problems we had when creating the algorithm and explain what appears to be inconsistent behavior from some of the data from our perspective.

Task Breakdown:

Our design was created entirely collaboratively using electronic tools or scheduled meeting times to work in tandem. Because of the nature of our project relative to the class, a lot of our progress gains were made conceptually while the code was fairly straightforward based off of this acquired knowledge. After the design was informally functioning as intended we began to work in parallel to complete the rest of the deliverables and tune the design. The method of displaying our design on a testbench was created by Chris, while the map of the Altera generated code was created by Isaac.

<u>Conclusion:</u>

In this project we created an audio effects unit which was able to perform 4 different effects to a stereo analog audio signal. We utilized signal conversion to allow the design to perform manipulations on digital audio. These manipulations were performed on magnitude values taken at a specific frequency of sample rate.

Overall, this project was very enjoyable to create though it did leave us extremely confused at many points. Our biggest hurdle ended up being the lack of documentation on the audio codec for this specific FPGA board. While it was fairly simple to get the board to pass analog audio in and out, it was not simple at all to access this data stream and perform manipulations upon it.We spent weeks investigating different methods for accessing and setting up the audio codecs, all with no success. The Altera IP cores seemed to be the best route to get the audio stream working, but the core itself would not integrate with the modules we wrote. We did manage to get audio streaming, but this turned out to be a simple audio passthrough mode that the board used when the Wolfson audio codec was not configured properly. We also attempted using Intel Platform Designer to integrate the design, going as far as making our own Platform Designer code block. This interface was confusing, and left us with more questions than answers.

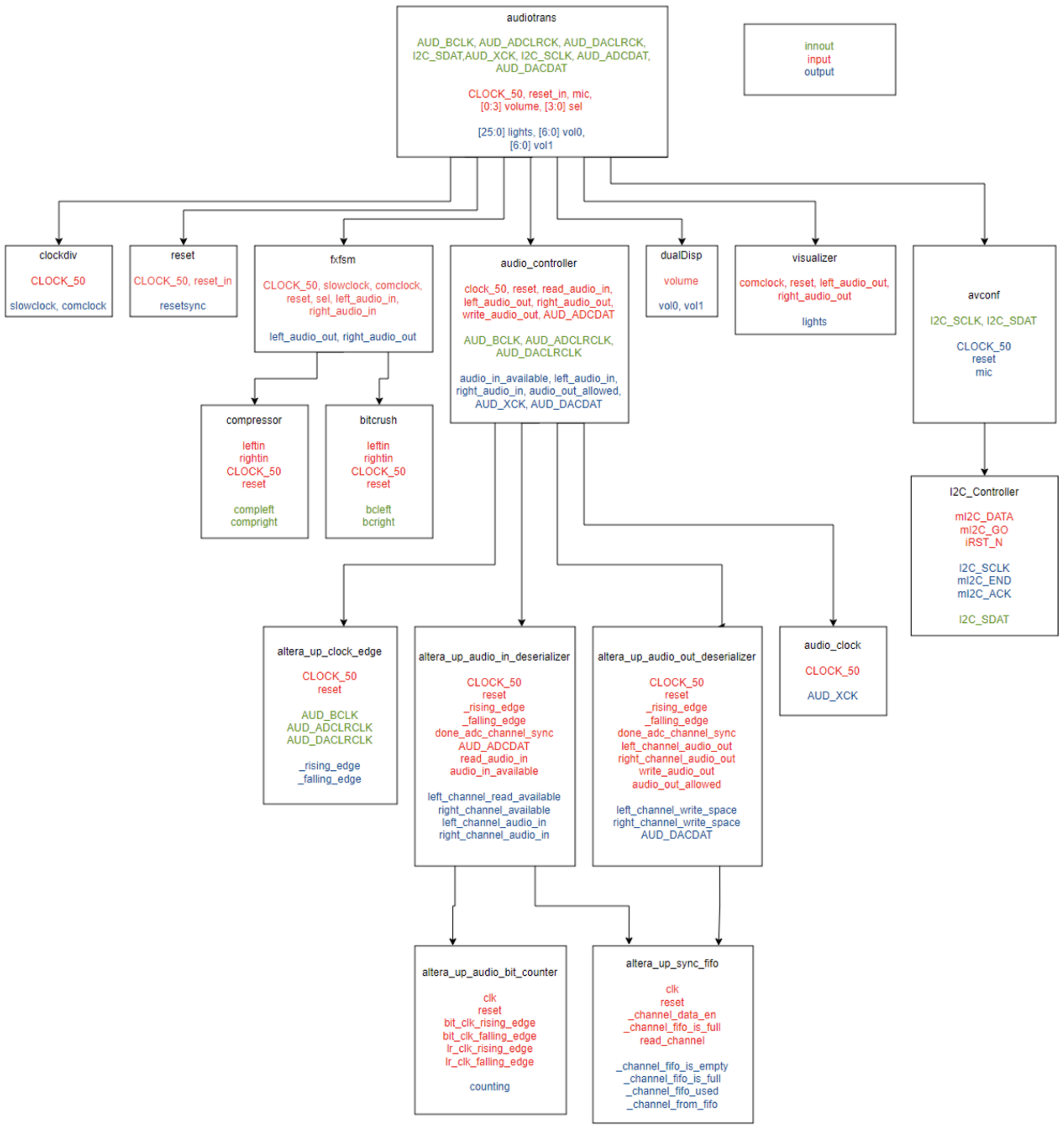
It wasn't until we encountered a website made by a student from University of Toronto that we were able to actually use or manipulate the data coming into the board. We did end up using the code packages from their website, but most of the code provided was actually from the IP cores from Intel, with the exception of a file named "avconf.v". This file would prove to be the missing link in our journey to get audio data into and out of the board. We discovered early on that the audio codec had to be manually configured via I2C to stream audio. We believed that the Platform Designer and IP cores automatically configured this codec for the developer since there was no mention of this required configuration or configuration process in any documentation online.

Since we did not want to just re-use their code, we opted to re-write the file in SystemVerilog and remove unnecessary clutter, like the video configuration data. This avoids the need for configuring the Avalon Data Bus, which is highly restrictive and irritatingly complex to use and understand. Instead, the audio codec is configured for use natively on the board.

This project led us down many bumpy roads, but ultimately left us with a plethora of useful knowledge about Altera boards, FPGA's, audio codecs, and digital audio processing.

Appendix and Reference:

Flow Chart of Entire Design Hierarchy



University of Toronto Student's Website

https://www.eecg.utoronto.ca/~jayar/ece241_08F/AudioVideoCores/