Axi4Lite Audio Equalizer

Ben Guan
Muyuan Li

Acknowledgement:

We would like to thank Professor Richter for your support and help throughout the semester.

“This is the most advanced and interesting proejct that I have worked on in my education career as a hardware engineer. I have to say this is by far my best learning experience at WashU. I am very glad that I took this class this semester.” - Ben

Table of Content:

1. Abstract

A 13-band audio equalizer is implemented in FPGA during the course of the fall semester. Audio input which is provided by a function generator is digitized using an Analog-to-Digital converter (ADC), the raw datas are sent to the Finite Impulse Response (FIR) module through Axi4Lite bus which calculates the discrete convolution of the sampled data and all 13 filter coefficients. The output is attenuated based on predefined scaling factors. The filtered datas are then converted to analog signals using a Digital-to-Analog converter (DAC). As shown in the figure below, we started from modeling the FIR filter in C, then modified the FIR filter verilog module to accommodate 13 filters. A testbench is used to test the functional correctness of the verilog code through simulation. Then we synthesized the code and instantiated the equalizer module as a IP block in the block diagram. A SDK program is used to verify the functional correctness of the hardware implementation by looking at the input and output waveform on the oscilloscope.

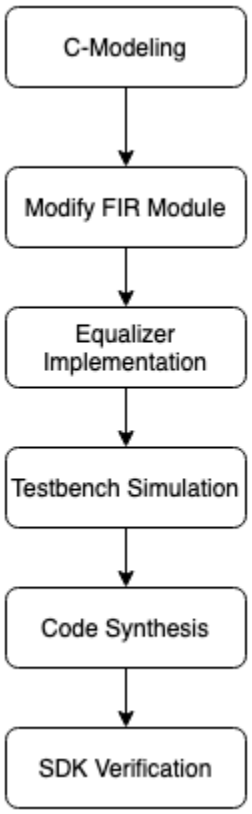


Figure 1. Flow Chart of Final Project

2. Introduction

The audio equalizer consists of 13 different filters: 1 low-pass filter, 11 band-pass filters, and 1 high-pass filter. All thirteen filter coefficients are loaded into the rams in the FIR module at the start of the program. Each ram stores 279 coefficients.

The figure below shows the frequency response of 13 different filters which includes a low-pass filter, a high-pass filter, and 11 band-pass filters. It can sample signals of 0 Hz to 25kHz which is a reasonable range for what the human ear is capable of hearing.

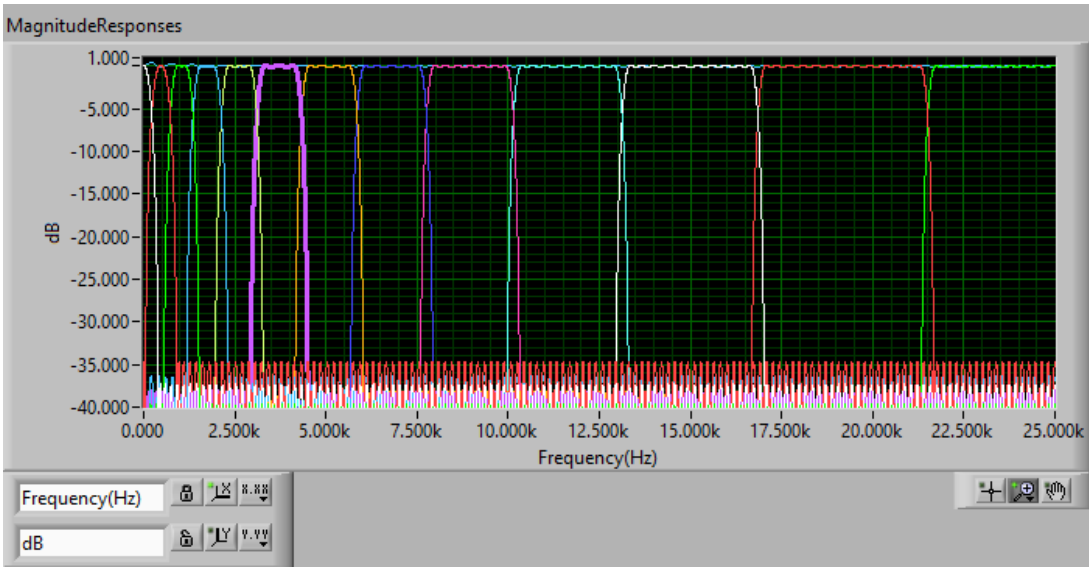


Figure 2. Frequency Response

As shown below, the light blue line is the aggregate frequency response of all 13 filters. We picked several frequency signals for verification purposes: 195Hz, 450Hz, and 750Hz.

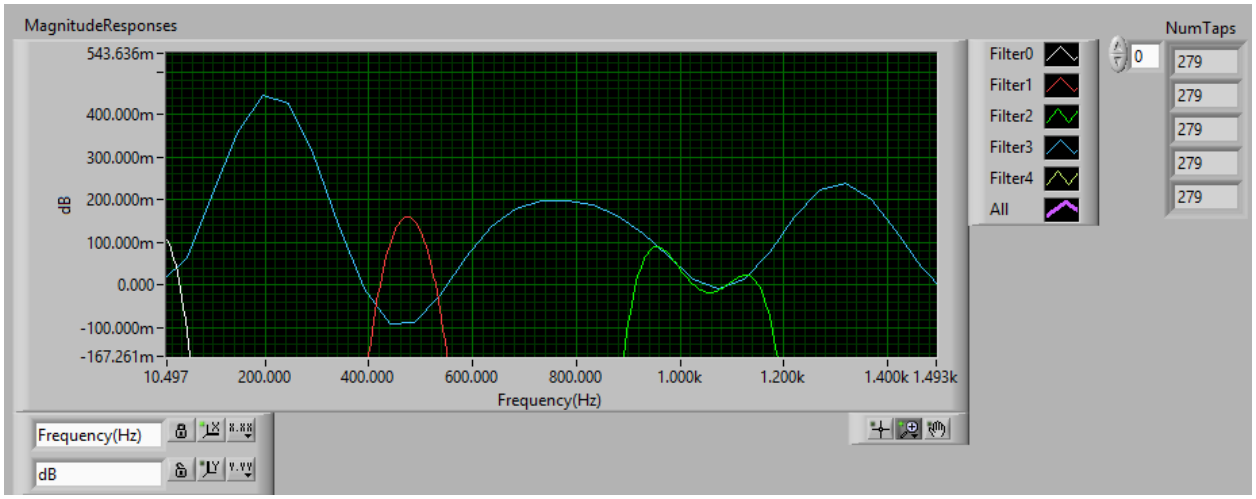


Figure 3. Close View of Frequency Response

The ADC has a data width of 16 bits. Based on the data sheet, the ADC uses a multiplexer for channel selection. we will set the first two bits of SDI to be “10” for channel 0 and “11” for channel 1. This pattern will alternate so each channel will receive samples at the same sampling rate. The CONV signal is active low meaning that when it is 0, the SPI is on and ready to transmit data in and out.

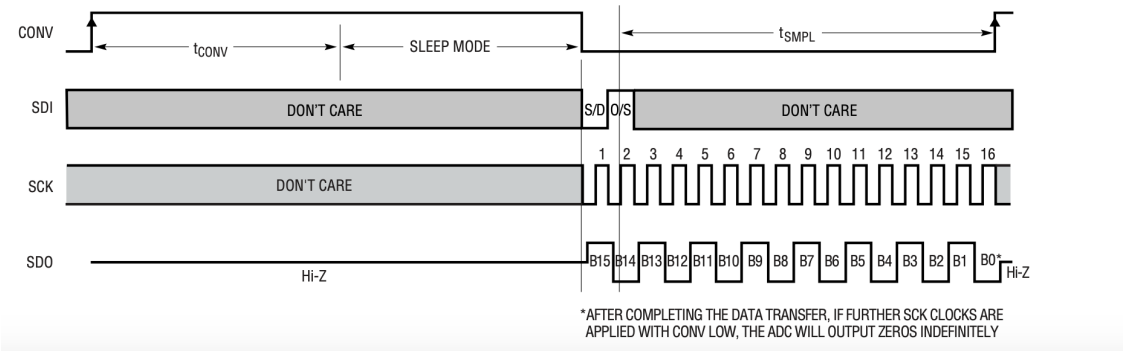


Figure 4. Timing Diagram of ADC

	MUX ADDRESS		CHANNEL #		
	SGL/DIFF	ODD/SIGN	0	1	GND
SINGLE-ENDED	1	0	+		–
MUX MODE	1	1		+	–
DIFFERENTIAL	0	0	+	–	
MUX MODE	0	1	–	+	

1864 TBL1

Table 1. Multiplexer Channel Selection

The DAC has a data width of 24 bits. There are 4 control bits, 4 address bits, 14 data bits, and 2 don’t care bits. Since we will be using single-ended mode, to load in data to channel 0, we output 0011 0000 + DATA to the DAC. To load in data to channel 1, we output 0011 0001 + DATA.

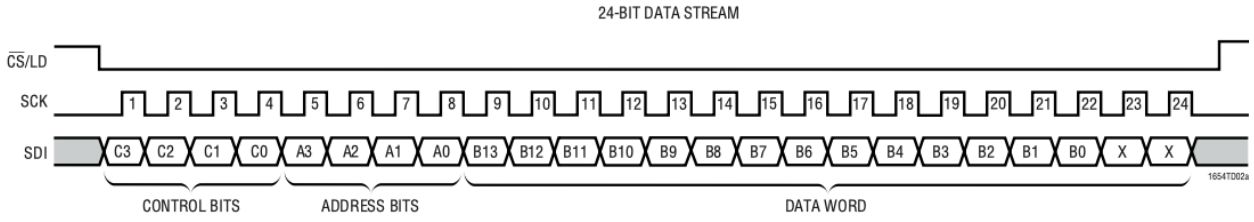


Figure 5. Timing Diagram of DAC

3.Design

3.1 Modified FIR Design

There are 6 states in the FIR peripheral: INIT, READY, CALC, CALC1, CALC2, and DATA_VALID. In INIT state, all of the filter coefficients are written to the peripheral and two circular buffers are zeroed out (one for each channel).

In READY state, xIndex and hIndex are reset. When we get a sample from the microblaze or testbench, FIR goes into CALC state which computes the convolution of the circular buffer and each individual filter. Since there are 279 taps in each filter, this state 279 clock cycles. In CALC1 state, the attenuated output is calculated by multiplying with the attenuation factor. In CACL2 state, all 13 attenuated outputs are summed up. DATA_VALID, we set the rdDataS which signals that the calculation is done and data is ready. The microblaze/testbench will constantly poll the flag until it is set to 1. Then in the next clock cycle, the data will be read.

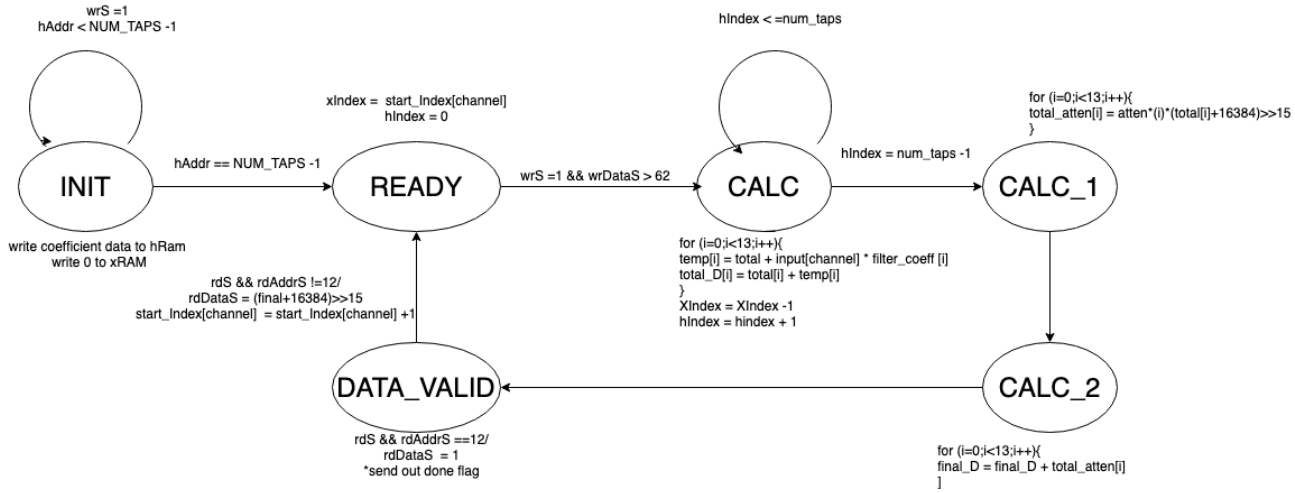


Figure 6: FSM of Axi4Lite FIR Peripheral

3.2 Equalizer FSM Design

As shown below, the FSM for the equalizer consists of 8 different states: IDLE, WRITE_ADC, READ_ADC1, READ_ADC2, FIR_CACL, FIR_WAIT, FIR_READ, and WRITE_DAC. In IDLE state, the microblaze/testbench writes data width, num_cycles, num_taps, all of the filter coefficients, as well as enabling all of the peripherals.

In IDLE state, ADC, DAC, and filter will be set up with required information, including the SDI data widths and operation cycle of required sample rate for ADC and DAC, and

number of taps along with value of taps for the filter. The state transitions into WRITE_ADC when an enable message is passed in.

In WRITE_ADC state, the channel will be set by sending either 0x8000 or 0xC0000 to ADC. When the writing is done, the state transitions into READ_ADC1. In READ_ADC1 state, a special flag will be continuously read from ADC. The flag will be 1 when ADC has the full SDO value recorded. The state will also check if the SDI that has just been sent in WRITE_ADC state is the first value passed in to both channels, if so ill will jump to WRITE_ADC state to avoid invalid SDO reading, otherwise, the state will transition into READ_ADC2.In READ_ADC2 state, the SDO value that ADC recorded will be polled. The state transitions into FIR_Calc.

In FIR_Calc state, the value that is just polled from ADC will be sent to the filter to generate a filtered output. When the writing process is done, the state will transition into FIR_Wait. In FIR_Wait state, a special flag will be continuously read from the filter. The flag will be 1 when calculation in the filter is done, the state will then transition to FIR_READ.In FIR_READ state, the filtered output value will be recorded and the state will transition into WRITE_DAC. In WRITE_DAC state, the control bit 0011, the channel 0001 or 0000, and the filtered output value will be sent to the DAC. The state will transition into WRITE_ADC.

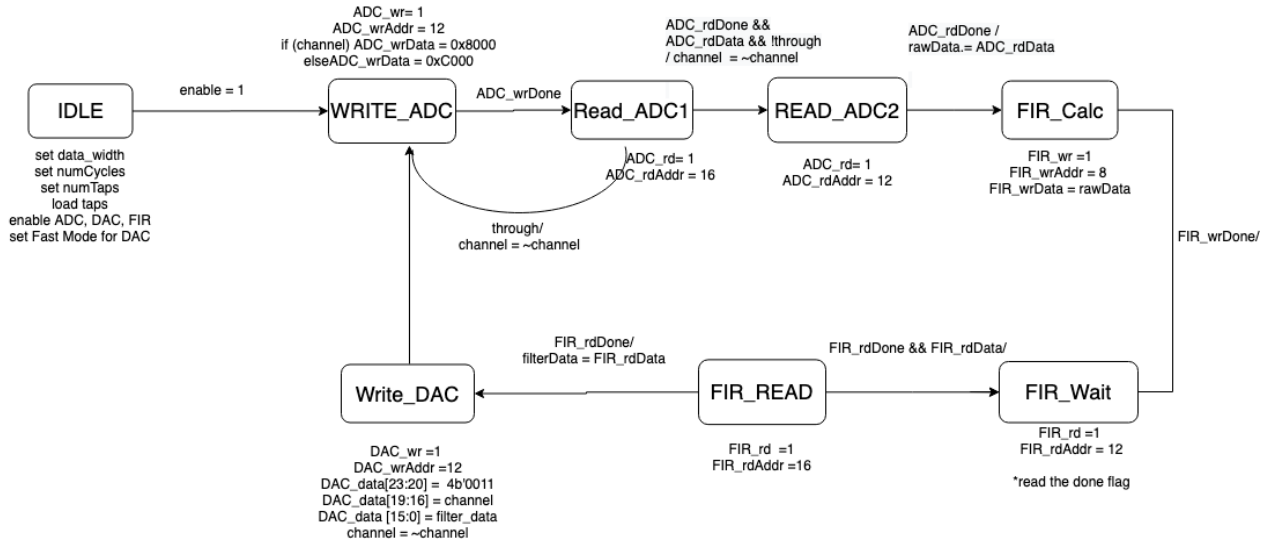


Figure 7: FSM of Axi4Lite Equalizer Peripheral

3.3 Equalizer System Design

As the figure shown below, the equalizer system includes an Axi4Lite supporter that communicates with the Axi4Lite manager that is connected to the microblaze. There the FSM controller which implements all of the state transitions in the equalizer.

Three managers are instantiated which “talk” to ADC_SPI, DAC_SPI, and FIR peripherals individually.The ADC SPI collects analog audio input from ADC which is connected to the frequency generator. The DAC SPI sends output to the DAC which is connected to an oscilloscope for output observation.

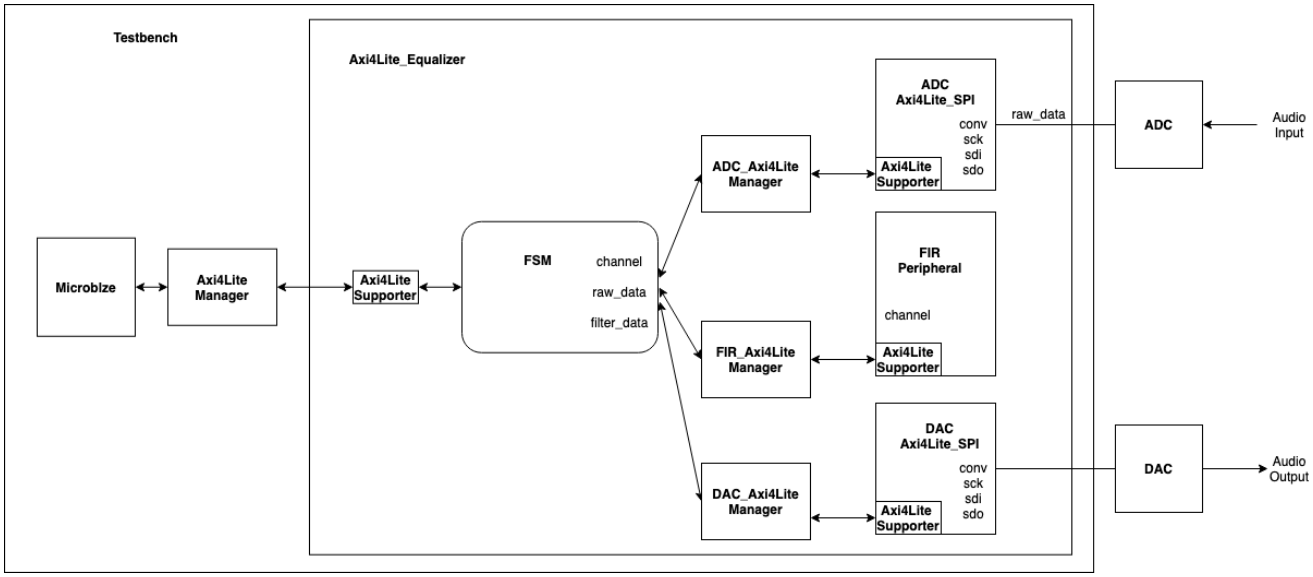


Figure 8: System Block Diagram of Audio Equalizer

4. Operation and Testing

4.1 C modeling:

We first modeled the 13-band FIR filter in C. The input samples are stored in a circular buffer. Then the convolution of input and each filter taps are calculated. The partial result is stored in Q1.15 format. The partial sum is also multiplied with the attenuation factor which is in Q2.14 format so that we can include an attenuation factor of 1 inclusive. All 13 partial sums are aggregated to get the total sum. The final result is converted to Q1.15 again. As shown in the figure below, the output signal is what we expected and the delay is about 139 samples, which is consistent with our expectation.

Delay = (N-1) /2 = (279 -1) /2 = 139

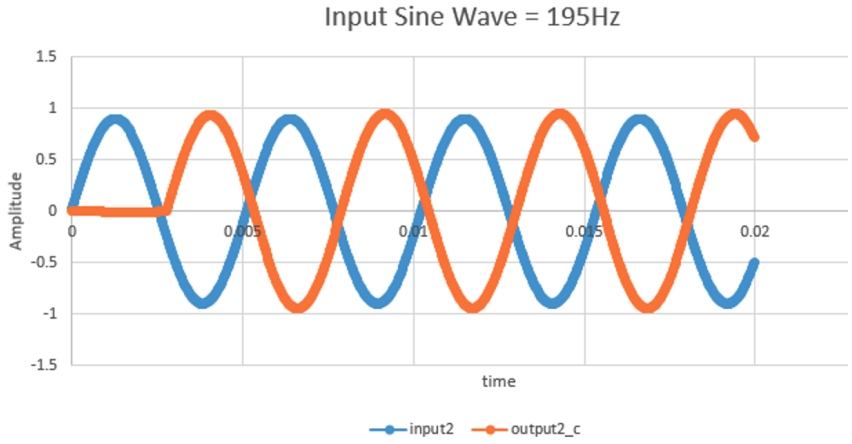


Figure 9: C-Simulated Output vs. Input for 195Hz Sine Wave

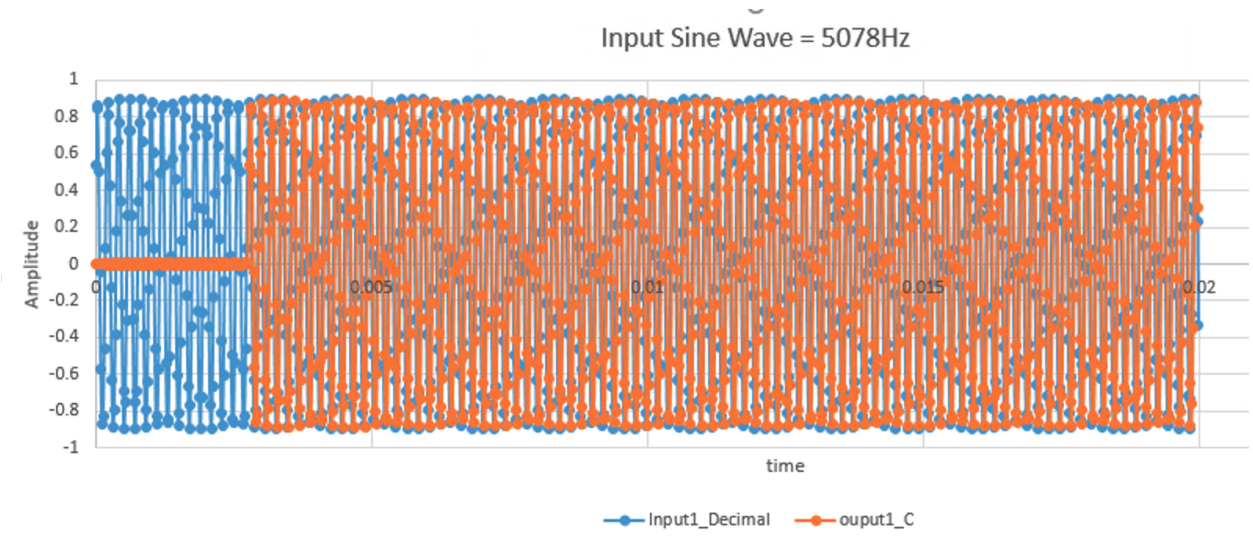


Figure 10: C-Simulated Output vs. Input for 195Hz Sine Wave

As shown in the table below, the expected gain for a 195Hz signal is about 444.9mdB. The gain calculated from the C-modeling is about 435.9mdB. Similarly, the expected gain for a 5078Hz signal is about -161.1mdB. The gain calculated from the C-modeling is about -168mdB. These calculated gains are consistent with the expected gain.

Sine Wave Frequency	195Hz	5078Hz
Gain in dB	0.4359	-0.1611
Expected Gain in dB	0.444.9	-0.1680

Table 2: C-Simulated Output vs. Input of 195Hz and 5078Hz Signal

4.2 Testbench Verification:

To test the functional correctness of the Axi4Lite Equalizer, we instantized an ADC tester and a DAC tester in the testbench which simulate the behavior of an ADC and a DAC, respectively.

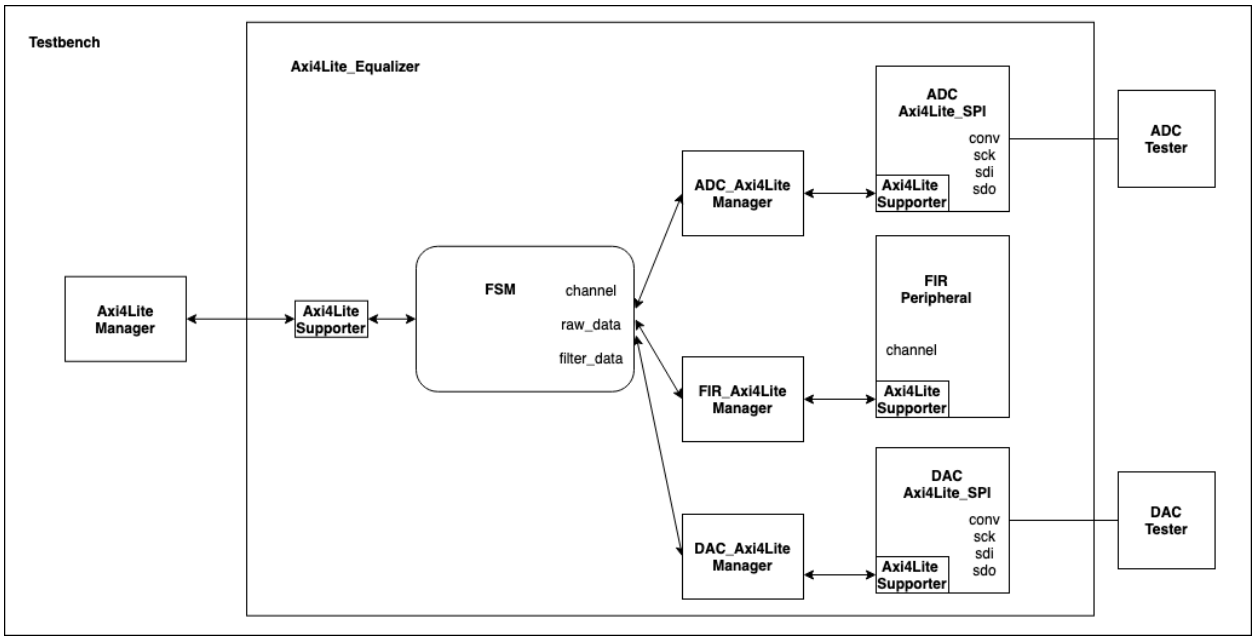


Figure 11: System Block Diagram of Audio Equalizer Simulation Testbench

The testbench code is shown below. It first sets up the data width and number of cycles in ADC and DAC. Then it sets up fast mode for the DAC. It also enables FIR and writes the number of taps to the corresponding pointer. It also writes all of the tap coefficients. Finally, it enables ADC, DAC, and audio equalizer.

```
initial begin
    $readmemh("taps.mem", Hmemory);

    M_AXI_ACLK =1;
    M_AXI_ARESETN = 0; //active low
    //initializing signals
    M_AXI_ARESETN = 0;
    M_AXI_ACLK = 0;
    rdAddr = 0;
    rd = 0;
    wrAddr = 0 ;
    wrData = 0 ;
    wr = 0;
    // Release reset
    #(CLK_PERIOD/2 + 2) M_AXI_ARESETN = 1;
    #(CLK_PERIOD*10)
   //enable equalizer

    //1. set up ADC
    write(4,16);    //Data width
    write(8,600);   //cycles

    //2. set up DAC
    write(16,24);     //Data width
    write(20,600);    //cycles
    write(28,'h50000000);  //set dac A fast mode
    write(28,'h51000000);  //set dac B fast mode

     //3. set up FIR
     write(32,1);     //enable FIR
     write(36,279); //write num taps

     $display(" loading hRam to FIR");

     for (i=0;i<3627;i=i+1)begin
         write(40,Hmemory[i]);
     end
     write(12,1);     //ADC enable
     write(24,1);     //DAC enable
     write(0,1);      //Equalizer enable
```

Figure 12. Testbench Initial Block Code

As shown in the figure below, it takes about 270us for all of the parameters to be sent to FIR, ADC_SPI, and DAC_SPI. It takes about 10 us for each data to be sampled and filtered. The DAC tester outputs all the data to the terminal, and that allows us to check if they are consistent with the output of the C-modeling.

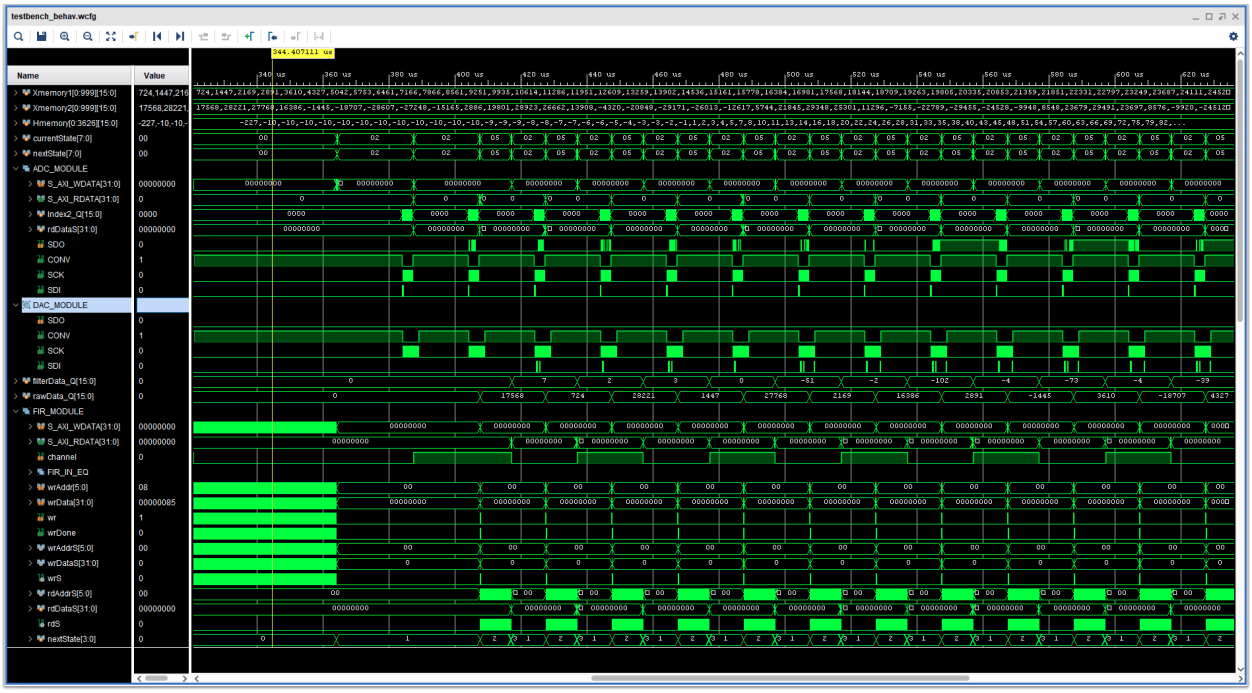


Figure 13. Vivado Simulation

4.3 Hardware Verification:

The equalizer module is instantiated as an IP block in the block diagram which has external ports connected to the ADC and DAC. The whole wrapper which includes the equalizer module, microblaze, and other peripherals are synthesized in Vavado. The bitstream is then exported. We wrote a C program in SDK which will run the actual hardware implementation.

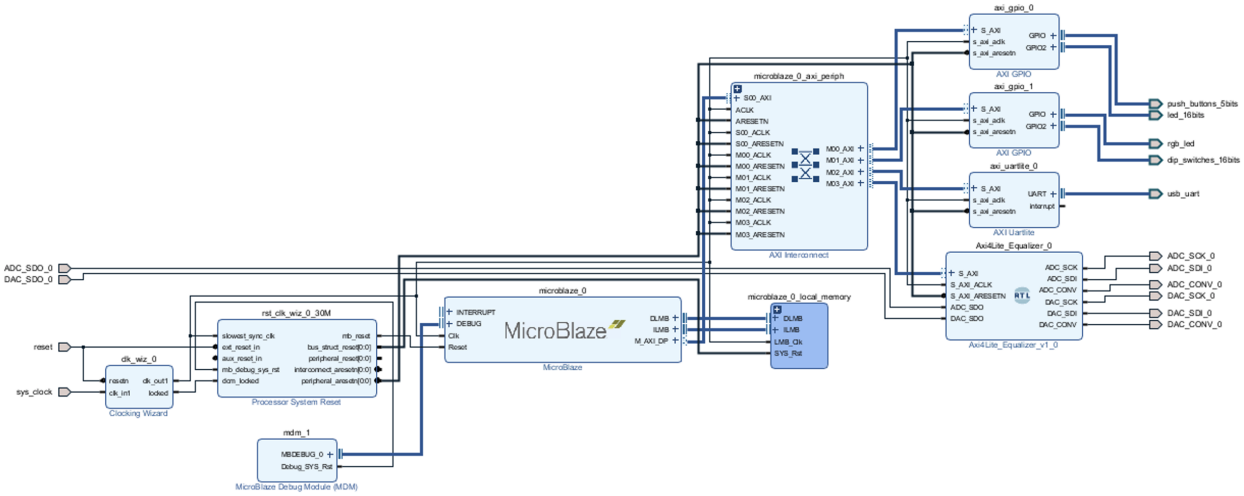


Figure 14. System Block Diagram of Audio Equalizer Implementation

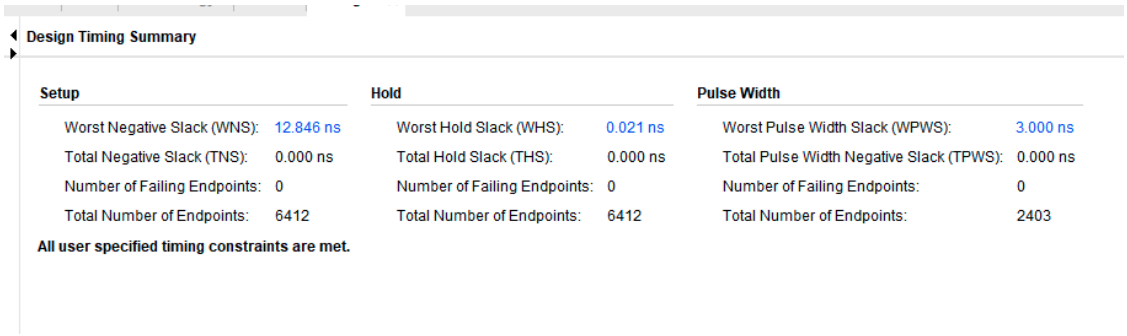
The SDK program is shown below. Similar to the function of verilog testbench, we first send all of the parameters including data width, number of cycles, number of taps per filter, as well as all of the filter coefficients to the equalizer module which will send the corresponding data to the corresponding modules based on the write address. Then we enable all of the modules.

```
volatile unsigned int *eqEnable  = (unsigned int *)  0x44a00000;    //0
volatile unsigned int *adcWidth   = (unsigned int *) 0x44a00004;    //4
volatile unsigned int *adcCycles  = (unsigned int *) 0x44a00008;    //8
volatile unsigned int *adcEnable  = (unsigned int *) 0x44a0000C;    //12

volatile unsigned int *dacWidth   = (unsigned int *) 0x44a00010;    //16
volatile unsigned int *dacCycles  = (unsigned int *) 0x44a00014;    //20
volatile unsigned int *dacEnable  = (unsigned int *) 0x44a00018;    //24
volatile unsigned int *dacData    = (unsigned int *) 0x44a0001C;    //28

volatile unsigned int *firEnable  = (unsigned int *) 0x44a00020;    //32
volatile unsigned int *numTaps  = (unsigned int *)   0x44a00024;        //36
volatile unsigned int *Taps  = (unsigned int *)      0x44a00028;        //40


int setup_FIR(){
    *firEnable = 1;
    *numTaps = 279;
    int i;
    for (i =0;i<3627;i++){
        *Taps = taps[i] ;
    }
}

int main(){
    print("running \n");
    *adcWidth = 16;     //ADC: 16 bits
    *adcCycles = 600;

    *dacWidth = 24;  //DAC: 24 bits
    *dacCycles = 600;

    *dacData = 0x500000; //set DAC A fast mode
    *dacData = 0x510000; //set DAC B fast mode

    setup_FIR();
    *adcEnable = 1;
    *dacEnable = 1;
    *eqEnable = 1;
    print("set up done \n");
    return 0;
}
```

Figure 15. SDK Program Code

5. Discussion

As shown in the figure below, the worst slack time and worst hold time are both positive.



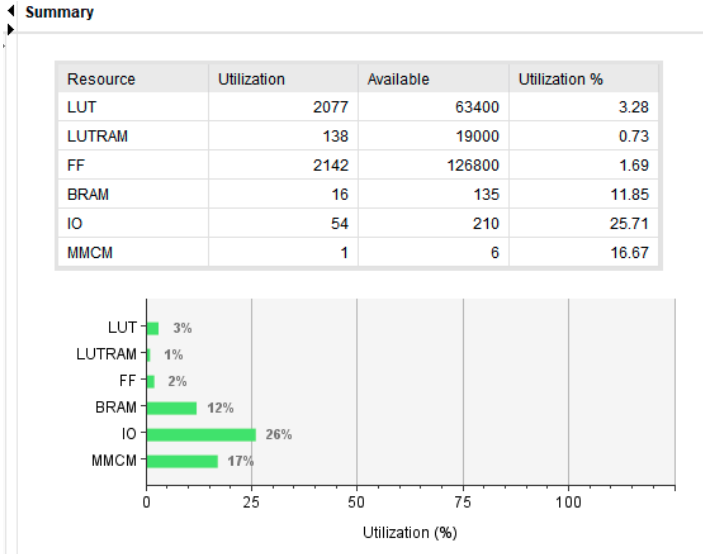
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	12.846 ns	Worst Hold Slack (WHS):	0.021 ns	Worst Pulse Width Slack (WPWS):	3.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	6412	Total Number of Endpoints:	6412	Total Number of Endpoints:	2403

All user specified timing constraints are met.

Figure 16: Timing Report

As shown in the figure below, the synthesized verilog code uses about 3.28% of total LUT and 1.69% of flips flops available on the DDR4 FPGA board.



Summary

Resource	Utilization	Available	Utilization %
LUT	2077	63400	3.28
LUTRAM	138	19000	0.73
FF	2142	126800	1.69
BRAM	16	135	11.85
IO	54	210	25.71
MMCM	1	6	16.67

Figure 17: Resource Utilization Summary Report

The synthesized digital circuit used about 0.229w on-chip power, which is smaller than what we expect.

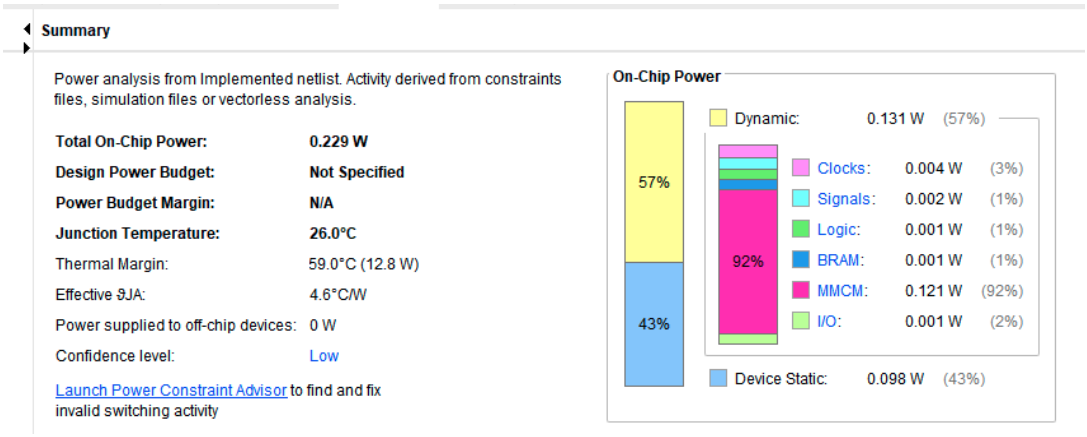


Figure 18. Power Summary Report

Figure below shows the input and output of a 195Hz Frequency. It shows a peak voltage of 2.24V.

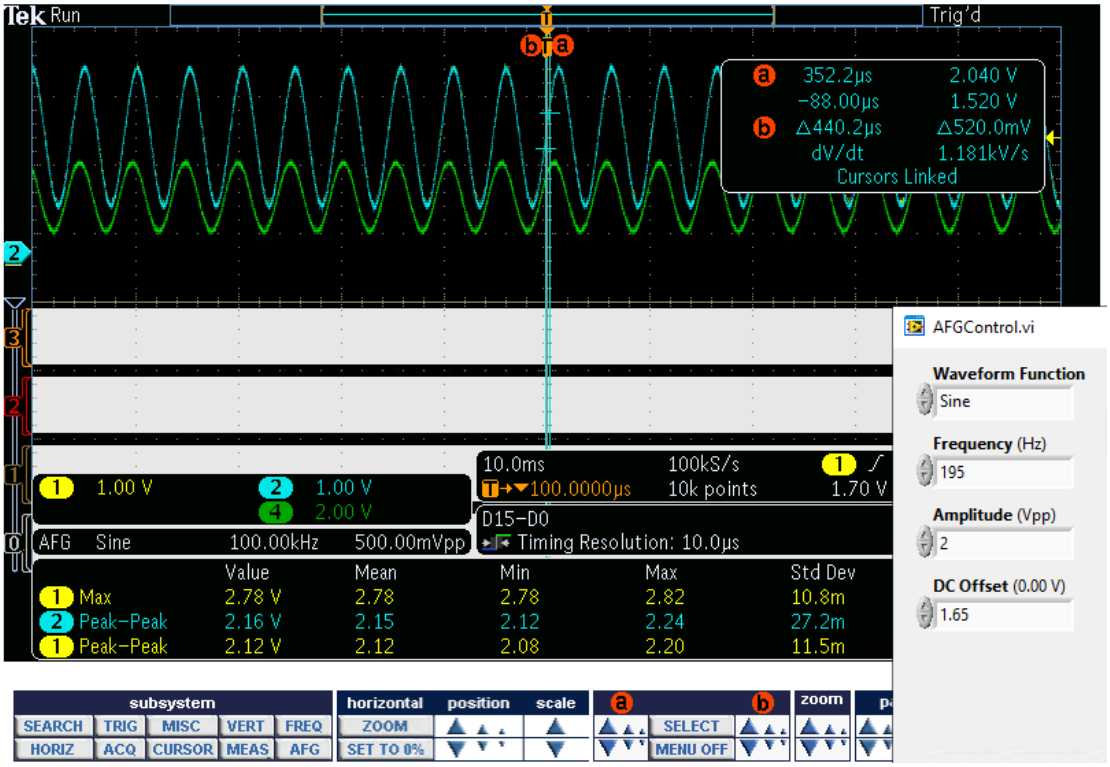


Figure 19. Oscilloscope Display of 195Hz Frequency

Figure below shows the input and output of a 450Hz Frequency. Since the max output is 2.12V and the max output for 195Hz input is 2.24V.

The difference in dB is 20*log(2.24/2.12)=478 mdB. We are expecting about 520mdB.

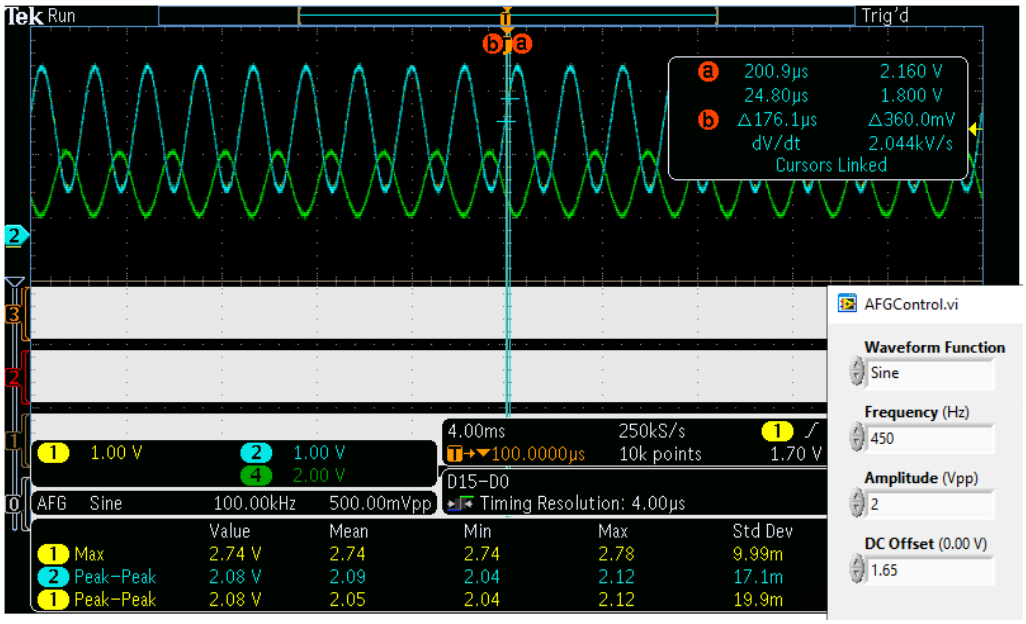


Figure 20. Oscilloscope Display of 450Hz Frequency

Similarity, for 750Hz input, the max output is 2.2V. The difference in dB compared to the 195Hz frequency is 20*log(2.24/2.2)=156 mdB, and we are expecting 220mdB difference. Thus, we see a consistent 40-60mdB discrepancy across several frequency resposne. Overall, we think that the discrepancy can be justifiable due the quantization noise and other hardware defects.

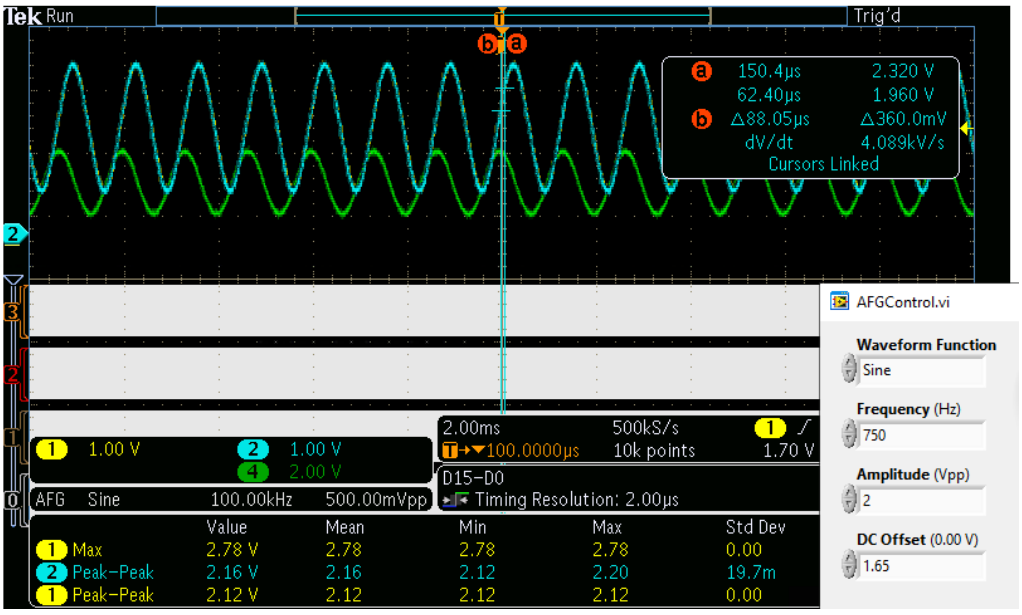


Figure 21. Oscilloscope Display of 750Hz Frequency

6. Conclusion

We have successfully implemented the system that can filter signals from two ADC channels and output the filtered signals to both DAC channels. After setting up all parameters in the ADC, FIR, and DAC modules, we can get data from the ADC, send it to FIR to calculate the discrete convolution, then get the filtered value back from FIR, and finally send the value to DAC to convert back to analog signals.

We use C to set up a model of the FIR, which is the key module of this project. By doing it, we can easily set up the filter in verilog and compare the verilog output with the C output.

We then use a test bench to check our implementation of FIR on verilog, before testing it in the equalizer module. We have to check that our design in calculating 13 filters is correct by comparing it with the C model result.

When wrapping up the equalizer system, we use another testbench to check the communication of all three modules. We also instantiated the ADC tester and DAC tester to simulate the functionalities of ADC and DAC. We found out that the ADC and DAC can only be enabled after FIR finishes its set up process. Otherwise we are not able to control the ADC and DAC in the period when FIR is setting up. We also find that the first value we read from ADC is not really valid, so we modify our design so that the FSM skips FIR and DAC for the first ADC write and read period.

Finally, we wrap everything up with the microblaze and use the SDK program to check our implementation. We check on the oscilloscope that the output matches the input on its amplitude and frequency. We also checked the gain difference for some peak and trough frequencies to verify that the result of our implementation is consistent with our expected differences. The discrepancy is around 40 -60 mdb which is acceptable. We conclude that the difference could be a result of quantization noise as well as ADC and DAC hardware defects. Overall, we successfully implemented a 13-band audio equalizer during the course of a semester. It was a great learning experience!