Bangladesh University of Engineering and Technology

Course No: EEE 6405

Course Title: Advanced VLSI Design

Assignment No: 1

Assignment Title: 8th-Order Finite Impulse Response(FIR) Filter Design And Hardware Implementation

Instructor:
Dr. A. B. M. Harun-Ur-Rashid Professor
Dept. of EEE, BUET

Date of Submission: 17.03.2023

Submitted by-
Name: Md Mahmudul Hasan
Department: EEE
Roll No.: 0421062325

FIR Filter Design And Implementation

Theory:

A FIR filter is a filter used in signal processing where the impulse response has a finite duration and settles to zero in a certain amount of time. An Nth-order discrete time FIR filter's impulse response lasts for exactly N+1 samples (from the first nonzero element to the last nonzero element) until it eventually zeros out. A direct form FIR filter's structure is shown in Figure 1. Each box with the label z-1 in this diagram represents a register cell with a one clock cycle delay.

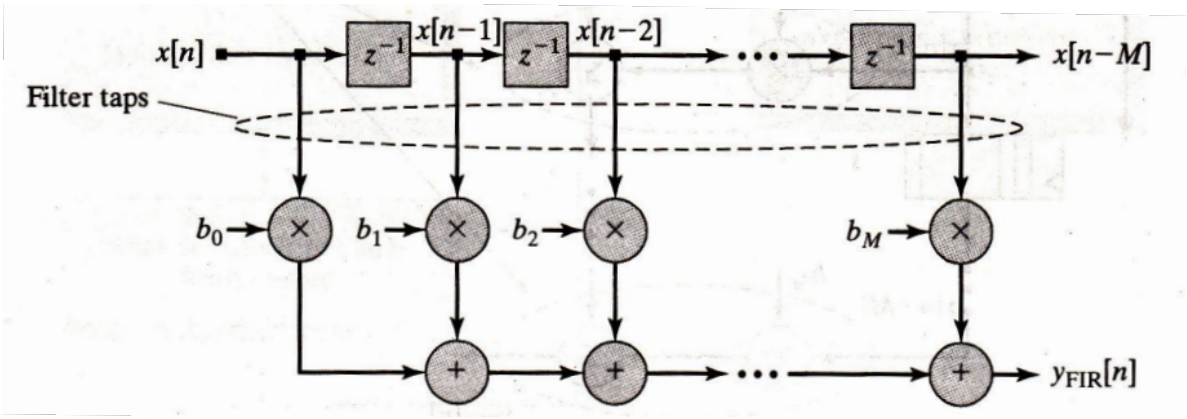


Figure 1: FIR filter direct form structure

The output sequence y[n] is expressed in terms of the input sequence x[n] using the equation:

$$y[n] = \sum_{i=0}^{N} b_i x[n - i]$$

where x[n] is the input signal, y[n] denotes the output signal, bi denotes the filter coefficients, and N denotes the filter order.

MATLAB Code For FIR Design:

A bandpass FIR filter of 8th order has designed using MATLAB with a sample rate of 48 kHz. cutoff frequency 8kHz & 16kHz. Tested using a sample analog signal of frequency within the passband and beyond the pass band. And final part converts the float coefficients of filter and test inputs of the filter to integer at the 8-bit binary format.

Code-

```
n = 8;
fs = 48000;
f1 = 8000;
f2 = 16000;
b = fir1(n,[2*f1/fs 2*f2/fs]);
freqz(b,1,[],fs);
subplot(2,1,1);
ylim([-30 10]);

% Generating random analog input signal with harmonics
fs = 48000;
dt = 1/fs;
StopTime = 2e-3; % Overall 2ms input signal
t = (0:dt:StopTime-dt)';
L = length(t);
Fc = fs/2;
a = 0.2;
x = 0.2*sin(2*pi*Fc*t*a)+0.4*sin(2*pi*2*Fc*t*a)+0.3*sin(2*pi*2*(Fc/4)*t*a);
x = abs(x);

% Plot the input signal versus time
figure;
plot(round(t*1e9),x);
xlabel('time (in nanoseconds)');
title('Signal versus Time');
```

```
% Plot frequency components of input signal
Y = fft(x);
P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
f = fs*(0:(L/2))/L;
figure;
plot(f,P1);
title('Single-Sided Amplitude Spectrum of X(t)');
xlabel('f (Hz)');
ylabel('|P1(f)|');


% Plot the output signal versus time
fx = filter(b,1,x);
figure;
plot(round(t*1e9),fx);
xlabel('time (in nanoseconds)');
title('Signal versus Time');


%Plot frequency components of filtered signal
Y = fft(fx);
P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
f = fs*(0:(L/2))/L;
figure;
plot(f,P1);
title('Single-Sided Amplitude Spectrum of X(t)');
xlabel('f (Hz)');
ylabel('|P1(f)|');

% Quantizing analog input in 8 bits to put as filter input
lx = min(x);
qx = (max(x)-lx)/255;

len = max(size(x));

for i=1:len
    x(i) = round((x(i)-lx)/qx);
end
disp('Input Signal:')
disp(dec2bin(x))
disp('         ')

%Quatized filter coeff
lb = min(b);
qb = (max(b)-lb)/255;

len = max(size(b));

for i=1:len
    b(i) = round((b(i)-lb)/qb);
end

disp('Filter Coefficient b:')
disp(dec2bin(b))
```

Console Output:

```
Input Signal:
00000000        11111010        11111111        01010100        00001010        01111111
11101000        01111010        01101010        10101100        00000000        10101100
01101010        01111010        11101000        01111111        00001010        01010100
11111111        11111010        00000000        11111010        11111111        01010100
00001010        01111111        11101000        01111010        01101010        10101100
00000000        10101100        01101010        01111010        11101000        01111111
00001010        01010100        11111111        11111010        00000000        11111010
11111111        01010100        00001010        01111111        11101000        01111010
01101010        10101100        00000000        10101100        01101010        01111010
11101000        01111111        00001010        01010100        11111111        11111010
00000000        11111010        11111111        01010100        00001010        01111111
11101000        01111010        01101010        10101100        00000000        10101100
01101010        01111010        11101000        01111111        00001010        01010100
11111111        11111010        00000000        11111010        11111111        01010100
```

```
00001010        01111111        11101000        01111010        01101010        10101100
00000000        10101100        01101010        01111010        11101000        01111111

Filter Coefficient b:
01010101        01001111        00000000        01001111        11111111        01001111
00000000        01001111        01010101
```

MATLAB Code Output Graph:

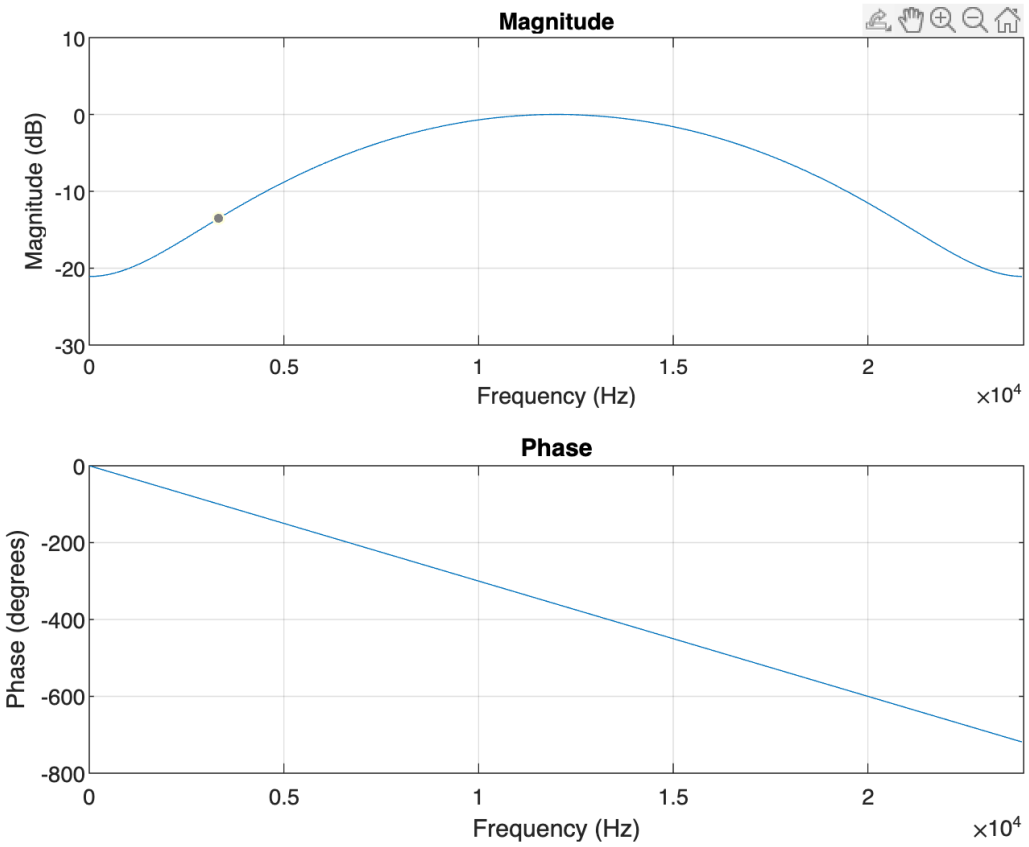


Figure 2: Filter Frequency & Phase Response

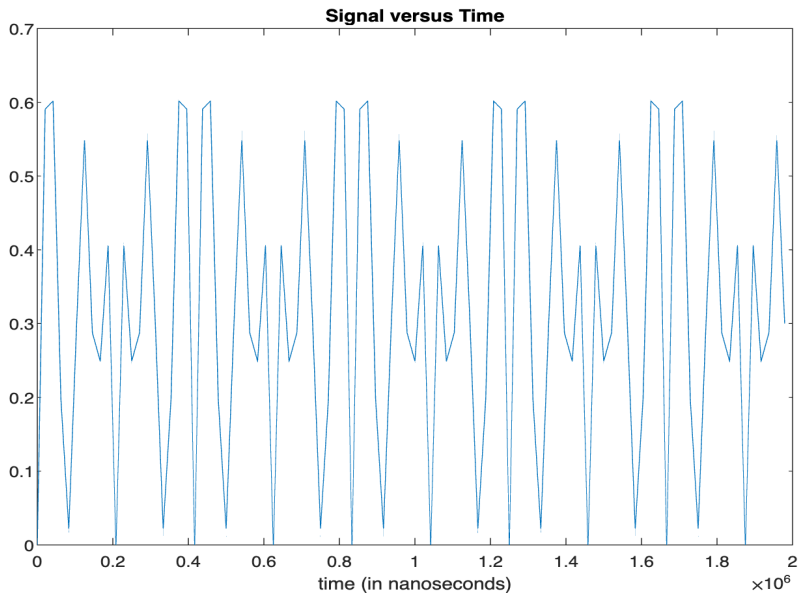


Figure 3: Input Test Signal

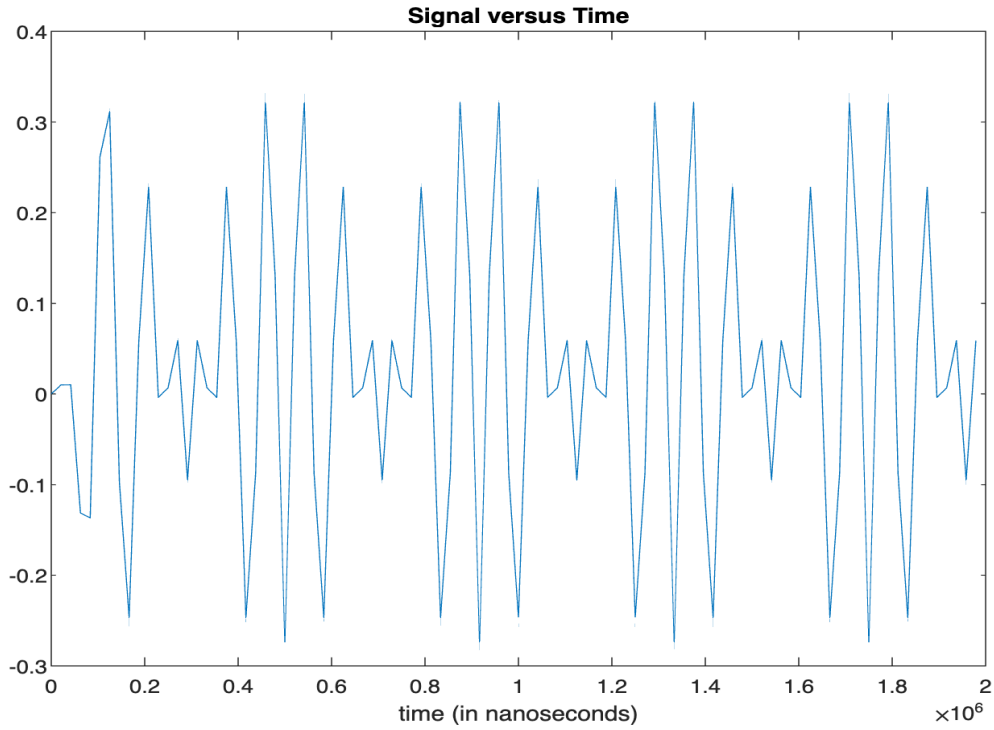


Figure 4: Filter Output of Test Signal

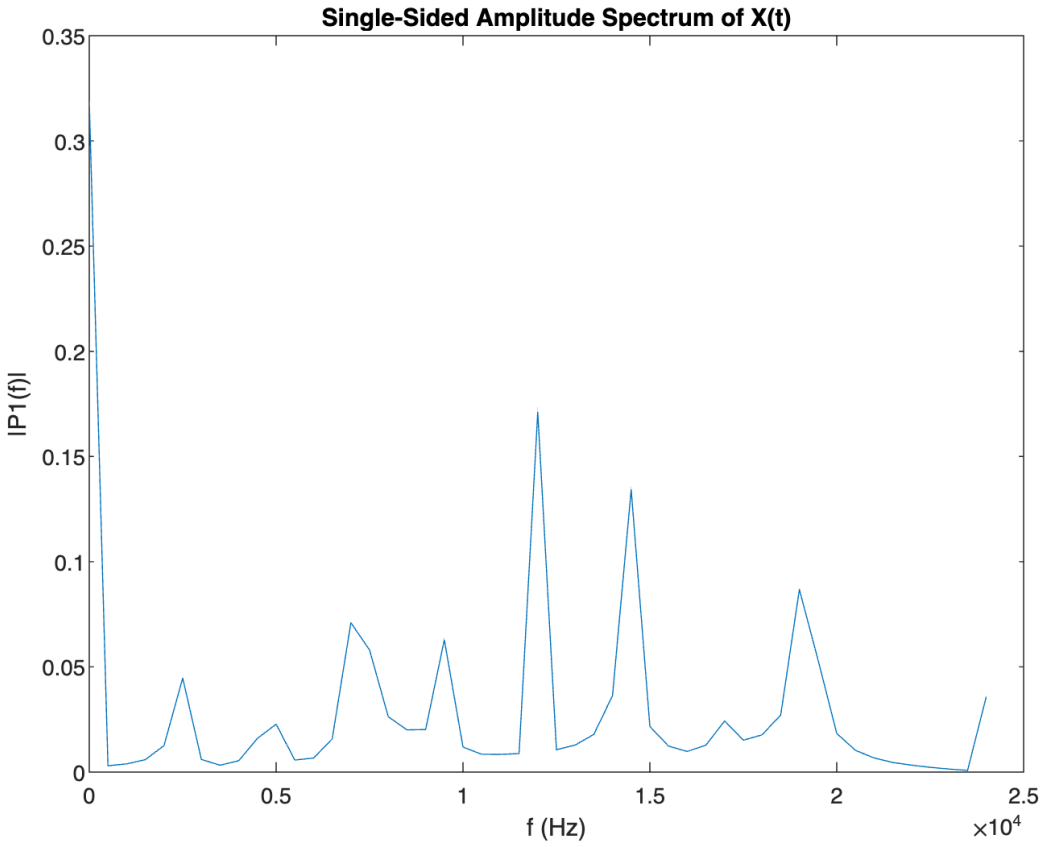


Figure 5: FFT of Filter Input Signal

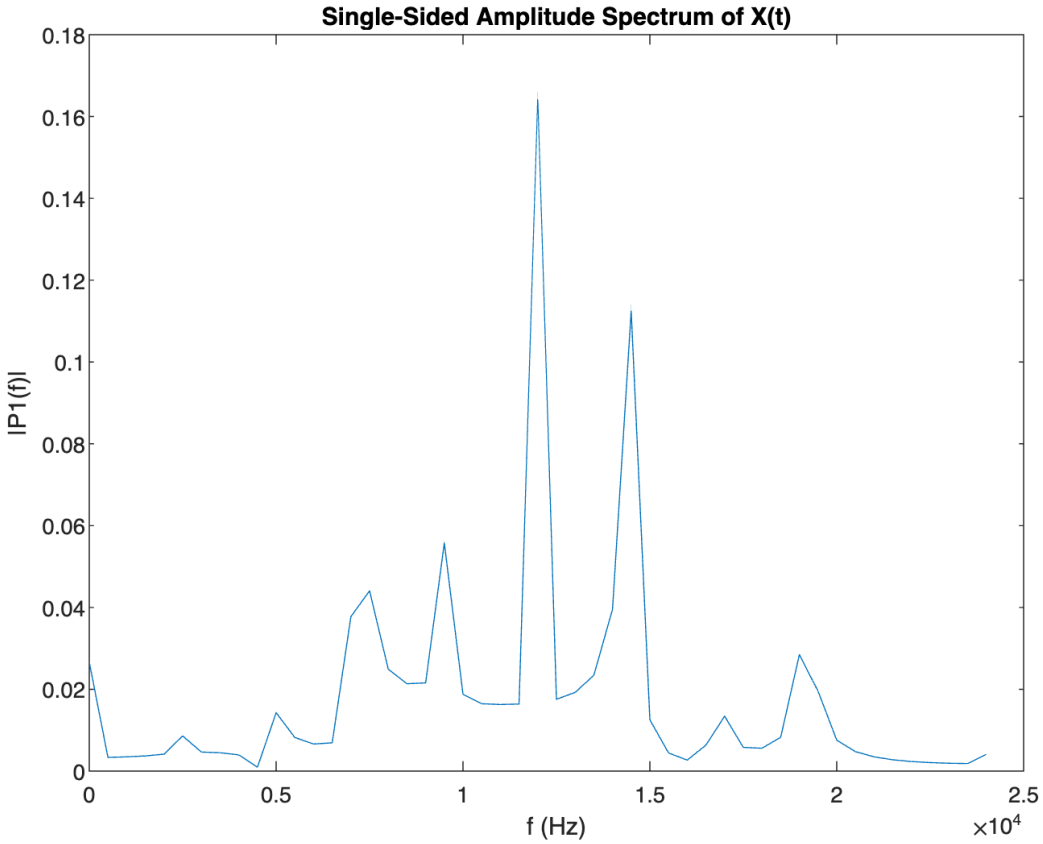


Figure 6: FFT of Filter Output Signal

Verilog Code For FIR:

Filter Module-

The following code module is an implementation of figure 1. And coefficients are inherited from the output of the matlab code.

```
module fir_filter (
    data_in, data_out, clk, rst
);
    output[17:0] data_out;
    input[7:0] data_in;
    input clk, rst;

    parameter b0 = 8'b01010101;
    parameter b1 = 8'b01001111;
    parameter b2 = 8'b00000000;
    parameter b3 = 8'b01001111;
    parameter b4 = 8'b11111111;
    parameter b5 = 8'b01001111;
    parameter b6 = 8'b00000000;
    parameter b7 = 8'b01001111;
    parameter b8 = 8'b01010101;
```

```
    reg[17:0] ff_data[1:8];

    assign data_out =
b0*data_in+b1*ff_data[1]+b2*ff_data[2]+b3*ff_data[3]+b4*ff_data[4]+b5*ff_data[5]+b6
*ff_data[6]+b7*ff_data[7]+b8*ff_data[8];

    integer k;
    always @ (posedge clk)
        if (rst==1)
        begin
            for (k=1;k<=8;k=k+1)
            begin
                ff_data[k]<=0;
            end
        end

        else
        begin
            ff_data[1]<=data_in;
            for (k=2;k<=8;k=k+1)
            begin
                ff_data[k]<=ff_data[k-1];
            end
        end

endmodule
```

Testbench Module-

The following code module is a testbench module for testing filter modules. Both FIR and IIR filter module has been tested using same module. For testing the input of filter has been inherited from the matlab output of input signal of the designed filter.

```
`timescale 1ns/10ps
module filter_testbench;

    reg[7:0] data_in;
    reg clk, rst;
    wire[17:0] data_out;


    fir_filter filter1(  //Chevychev_highpass instead of fir_filter for iir
        .data_in(data_in),
        .data_out(data_out),
        .clk(clk),
        .rst(rst)
    );


    initial
        begin
```

```
          clk = 1'b0;
      forever
          begin
              #10416 clk = ~clk;
              if (clk == 1)
                begin
                  $display(data_out);
                end
          end
       end

   initial
   begin
       #0      rst=1;
               data_in=8'b00000000;
       #20832  rst=0;
       #20833  data_in=8'b00000000;
       #20834  data_in=8'b11111010;
       #20833  data_in=8'b11111111;
       #20834  data_in=8'b01010100;
       #20833  data_in=8'b00001010;
       #20834  data_in=8'b01111111;
       #20833  data_in=8'b11101000;
       #20834  data_in=8'b01111010;
       #20833  data_in=8'b01101010;
       #20834  data_in=8'b10101100;
       #20833  data_in=8'b00000000;
       #20834  data_in=8'b10101100;
       #20833  data_in=8'b01101010;
       #20834  data_in=8'b01111010;
       #20833  data_in=8'b11101000;
       #20834  data_in=8'b01111111;
       #20833  data_in=8'b00001010;
       #20834  data_in=8'b01010100;
       #20833  data_in=8'b11111111;
       #20834  data_in=8'b11111010;
       #20833  data_in=8'b00000000;
       #20834  data_in=8'b11111010;
       #20833  data_in=8'b11111111;
       #20834  data_in=8'b01010100;
       #20833  data_in=8'b00001010;
       #20834  data_in=8'b01111111;
       #20833  data_in=8'b11101000;
       #20834  data_in=8'b01111010;
       #20833  data_in=8'b01101010;
       #20834  data_in=8'b10101100;
       #20833  data_in=8'b00000000;
       #20834  data_in=8'b10101100;
       #20833  data_in=8'b01101010;
       #20834  data_in=8'b01111010;
       #20833  data_in=8'b11101000;
```

```
        #20834  data_in=8'b01111111;
        #20833  data_in=8'b00001010;
        #20834  data_in=8'b01010100;
        #20833  data_in=8'b11111111;
        #20834  data_in=8'b11111010;
        #20833  data_in=8'b00000000;
        #20834  data_in=8'b11111010;
        #20833  data_in=8'b11111111;
        #20834  data_in=8'b01010100;
        #20833  data_in=8'b00001010;
        #20834  data_in=8'b01111111;
        #20833  data_in=8'b11101000;
        #20834  data_in=8'b01111010;
        #20833  data_in=8'b01101010;
        #20834  data_in=8'b10101100;
        #20833  data_in=8'b00000000;
        #20834  data_in=8'b10101100;
        #20833  data_in=8'b01101010;
        #20834  data_in=8'b01111010;
        #20833  data_in=8'b11101000;
        #20834  data_in=8'b01111111;
        #20833  data_in=8'b00001010;
        #20834  data_in=8'b01010100;
        #20833  data_in=8'b11111111;
        #20834  data_in=8'b11111010;
        #20833  data_in=8'b00000000;
        #20834  data_in=8'b11111010;
        #20833  data_in=8'b11111111;
        #20834  data_in=8'b01010100;
        #20833  data_in=8'b00001010;
        #20834  data_in=8'b01111111;
        #20833  data_in=8'b11101000;
        #20834  data_in=8'b01111010;
        #20833  data_in=8'b01101010;
        #20834  data_in=8'b10101100;
        #20833  data_in=8'b00000000;
        #20834  data_in=8'b10101100;
        #20833  data_in=8'b01101010;
        #20834  data_in=8'b01111010;
        #20833  data_in=8'b11101000;
        #20834  data_in=8'b01111111;
        #20833  data_in=8'b00001010;
        #20834  data_in=8'b01010100;
        #20833  data_in=8'b11111111;
        #20834  data_in=8'b11111010;
        #20833  data_in=8'b00000000;
        #20834  data_in=8'b11111010;
        #20833  data_in=8'b11111111;
        #20834  data_in=8'b01010100;
        #20833  data_in=8'b00001010;
        #20834  data_in=8'b01111111;
```

```
        #20833  data_in=8'b11101000;
        #20834  data_in=8'b01111010;
        #20833  data_in=8'b01101010;
        #20834  data_in=8'b10101100;
        #20833  data_in=8'b00000000;
        #20834  data_in=8'b10101100;
        #20833  data_in=8'b01101010;
        #20834  data_in=8'b01111010;
        #20833  data_in=8'b11101000;
        #20834  data_in=8'b01111111;
        #20832  $stop;
    end

    initial
    begin
        $dumpfile("filter_dump.vcd");
        $dumpvars;
    end
endmodule
```

Output Waveform In EDA-Playground:

The "filter_dump.vcd" file is dumped from the testbench when simulating in EDA playground. The output wave is shown in Figure 14.

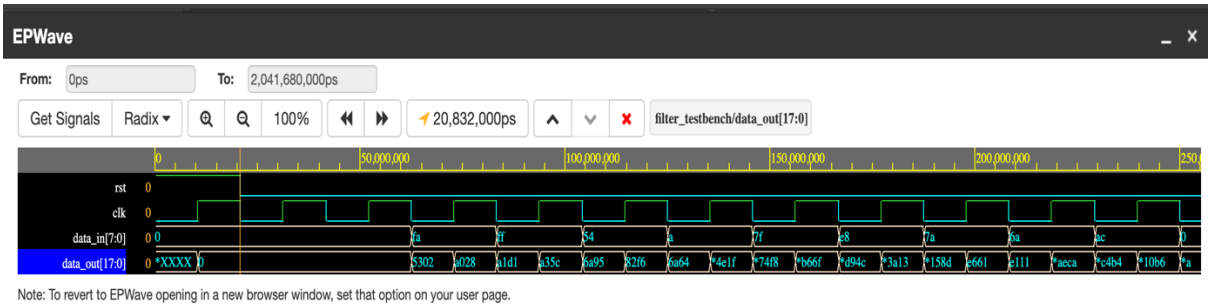


Figure 7: Simulation result waveform in EDA Playground

Filter output Analysis:

The output of Verilog file is exported to a .txt file and executed a comparison program in MATLAB by dequeuing the 18 bit digital signal to analog signal. Further it has been compared with the filter output given by the MATLAB for the same filter. The code for comparison is given below-

```
clc;
clear;
close all;

%%The 8th order bandpass FIR & IIR filter was designed using MATLAB with a sample
%%rate of 48 kHz. cutoff frequency 8kHz & 16kHz
n = 8;
fs = 48000;
f1 = 8000;
f2 = 16000;
b = fir1(n,[2*f1/fs 2*f2/fs]);

% Generating random analog input signal with harmonics
fs = 48000;
dt = 1/fs;
StopTime = 2e-3; % Overall 2ms input signal
```

```
t = (0:dt:StopTime-dt)';
L = length(t);
Fc = fs/2;
a = 0.2;
x = 0.2*sin(2*pi*Fc*t*a)+0.4*sin(2*pi*2*Fc*t*a)+0.3*sin(2*pi*2*(Fc/4)*t*a);
x = abs(x);
fx = filter(b,1,x);

%Read the output txt file
xf = load('xrun_fir.txt');
%xf = load('xrun_iir.txt')


% Deuantizing 18bit digital output to analogue signal
n = 18;
Lf = 2^n;
lx = min(fx);
qx = (max(fx)-lx)/Lf;
%these two coefficient are used for the amplitude change
% to compare with matlab output. the values are chosen by trial
% and error method
dq_adj_cof = 2;
dc_ofset = 0.7;
len = max(size(xf));

for i=1:len
    xf(i) = dq_adj_cof*(xf(i)*qx+dc_ofset*lx);
end

% Plot the output signal versus time
figure;
plot(xf,'color','b');
hold on;
plot(fx,'color','r');
legend('verilog', 'matlab')
xlabel('time (in nanoseconds)');
title('Signal versus Time');

%Plot frequency components of filtered signal
%for matlab output
Y = fft(fx);
P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
f = fs*(0:(L/2))/L;

%for verilog output
Yf = fft(xf);
P2f = abs(Yf/L);
P1f = P2f(1:L/2+1);
P1f(2:end-1) = 2*P1f(2:end-1);
f = fs*(0:(L/2))/L;
figure;
plot(f,P1,'color','r');
hold on;
plot(f,P1f,'color','b');
legend('matlab', 'verilog')
title('Single-Sided Amplitude Spectrum of X(t)');
xlabel('f (Hz)');
ylabel('|P1(f)|');
```

Output of the code is given below

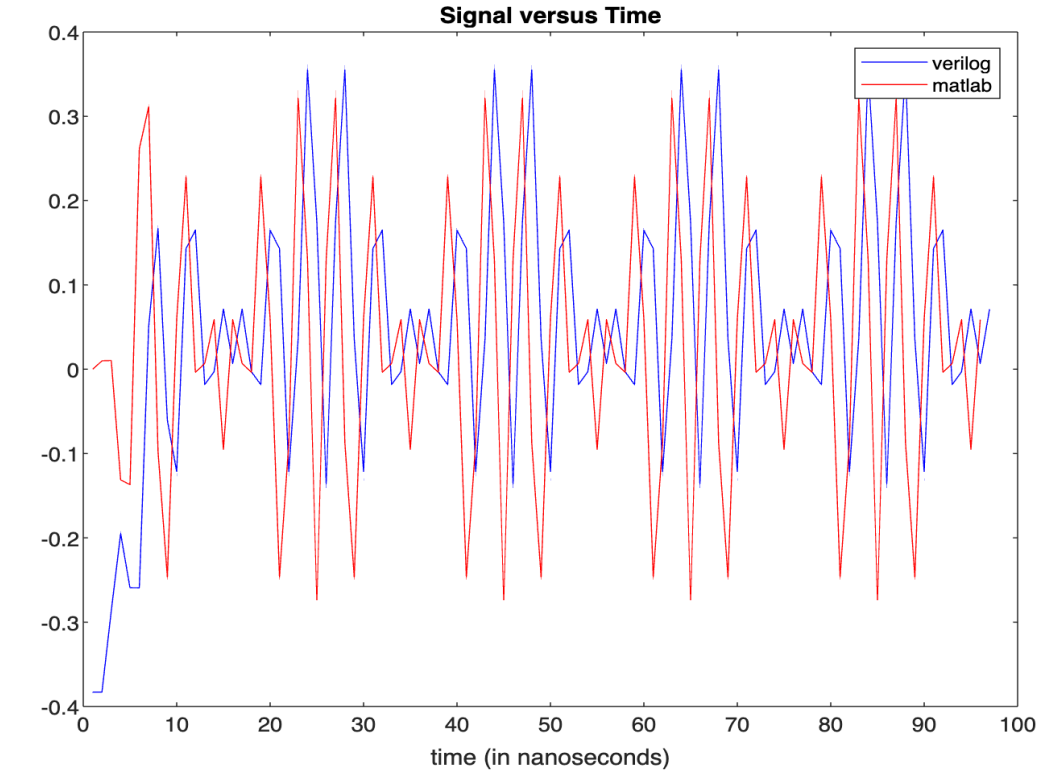


Figure 8: Filtered output comparison in time domain

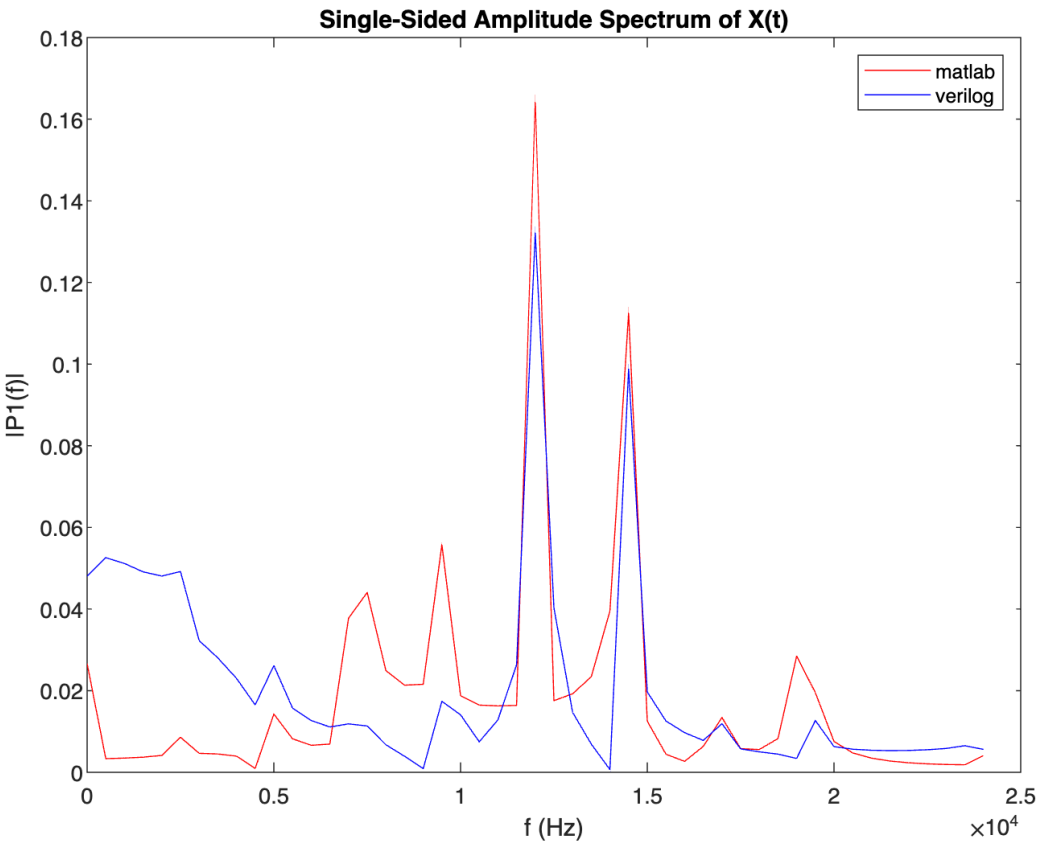


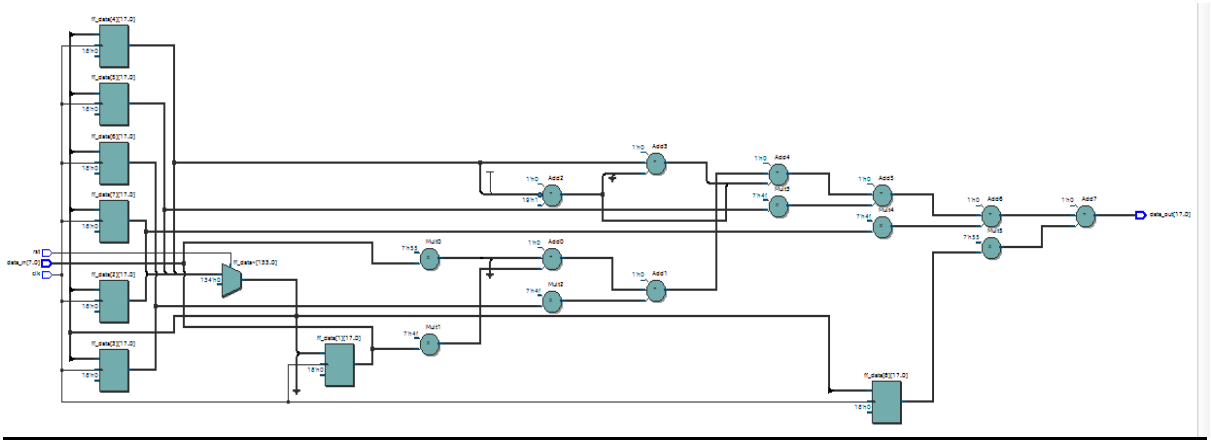
Figure 9: Filtered output comparison in frequency domain

Synthesize in Cyclone V by Quartus Prime Lite for FIR:

The design was compiled in a Cyclone V board and then synthesized. The summary of the synthesis is mentioned below-

Parameter	Value
FPGA Family	Cyclone V
Device	5CSEMA5F31C6
Logic utilization (in ALMs)	Used - 30 / Total- 32,070 (< 1 %)
Total registers	64
Total pins	28 / 457 (6 %)
Total DSP Blocks	5 / 87 (6 %)

The RTL of the design was generated which is shown as below.



Summary:

The dequantization portion can be integrated in the design Verilog module. It has been done here in MATLAB to easily find out the correct dequantization factor which would divide the output of the FIR hardware. Fixed point arithmetic is used here as it is implemented in an FPGA which requires the design to be synthesizable. Floating point arithmetic using IEEE 754 double precision format would show more precise result but the design would not be synthesizable nor applicable in FPGA.