Bangladesh University of Engineering and Technology

Course No: EEE 6405

Course Title: Advanced VLSI Design

Assignment No: 2

Assignment Title: 9th-Order Infinite Impulse Response(IIR) Filter Design And Hardware Implementation

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

Date of Submission: 15.03.2023

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

Theory:

The output from the infinite impulse response (IIR) filter is generated utilizing the current and prior inputs as well as past outputs, making it a recursive filter. There is feedback from the output in the filter structure because the filter uses earlier values of the output. A direct form 1 IIR filter's construction is shown in the following figure. Each box with the designation z-1 in this diagram represents a register cell with a one clock cycle delay.

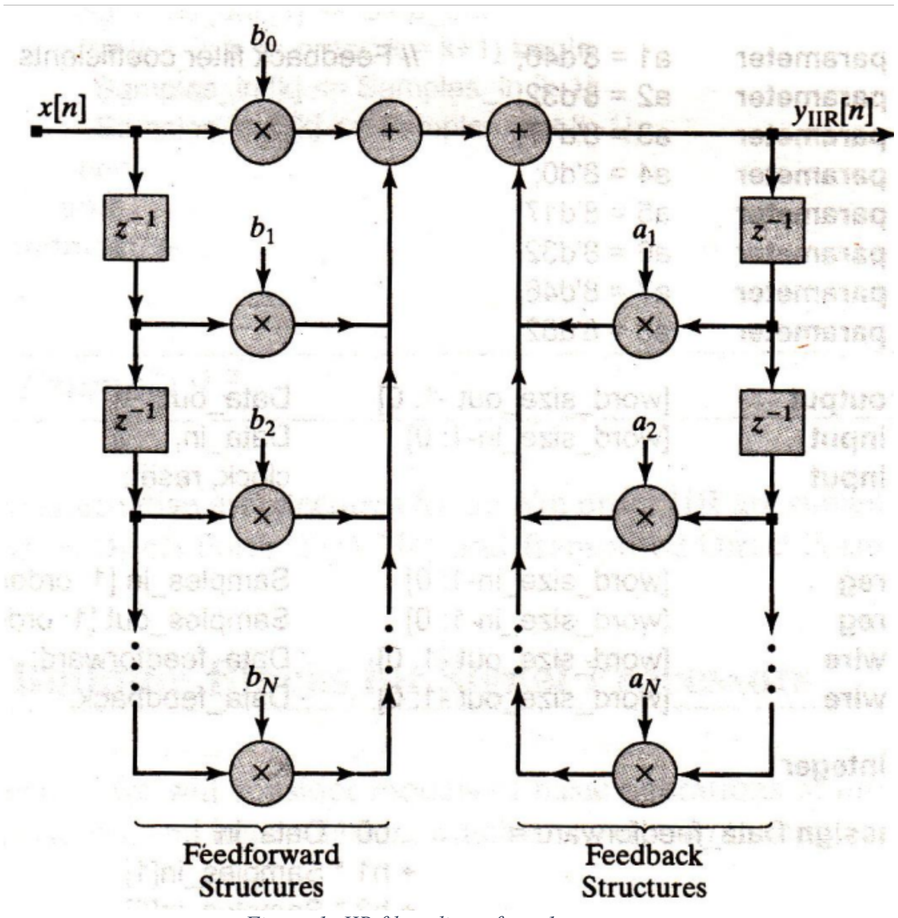


Figure 1: IIR filter direct form 1 structure

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

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

where x[n] is the input signal, y[n] denotes the output signal, b_i denotes filter's numerator/feedforward coefficients, a_j are the filter's denominator/feedback coefficients. M denotes the filter order and N specifies how many prior samples of the input will be used to form the output.

MATLAB Code For IIR Design:

A high-pass IIR filter of 9th order has designed using MATLAB with a sample rate of 48 kHz. cutoff frequency is 8kHz and passband ripple is 0.5dB. 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:

```
clc;
clear;
close all;

%%The 9th order highpass IIR filter was designed using MATLAB with a sample
%%rate of 48 kHz. cutoff frequency 8kHz
n=9;
rp=0.5;
fs = 48000;
```

```matlab
f1 = 8000;
[b,a] = cheby1(n,rp,2*f1/fs,'high');
freqz(b,a,[],fs);

% Generating 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;
fac = 0.2;
x = 0.2*sin(2*pi*Fc*t*fac)+0.4*sin(2*pi*2*Fc*t*fac)+0.3*sin(2*pi*2*(Fc/4)*t*fac);
%x = 0.2*sin(2*pi*Fc*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,a,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 n bits to put as filter input
n_bit = 8;
L = 2^n_bit;
lx = min(x);
qx = (max(x)-lx)/L;
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 n bit
n_bit=8;
L=2^n_bit;
lb = min(b);
qb = (max(b)-lb)/L;
len = max(size(b));
for i=1:len
    b(i) = round((b(i)-lb)/qb);
end
disp('feedforward coefficients, b:')
disp(dec2bin(b))
disp('         ')
```

```
la = min(a);
qa = (max(a)-la)/L;
len = max(size(a));
for i=1:len
    a(i) = round((a(i)-la)/qa);
end
disp('feedback coefficients, a:')
disp(dec2bin(a))
disp('          ')
```

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

feedforward coefficients, b:
10000001        01110110        10100100        00101011        11111111        00000000
11010100        01011011        10001001        01111110

feedback coefficients, a:
10011101        00000000        11111111        00010010        10111110        01011101
01110011        01100100        01010101        01010101
```

MATLAB Code Output Graph:

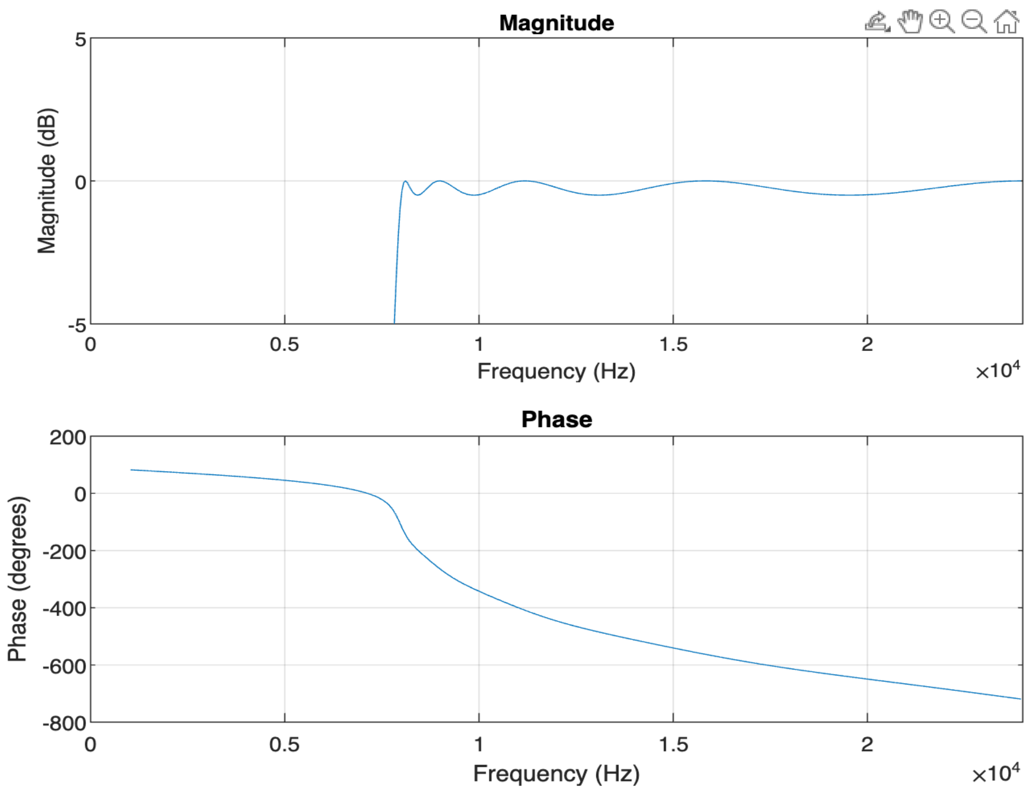


Figure 2: Filter Frequency & Phase Response

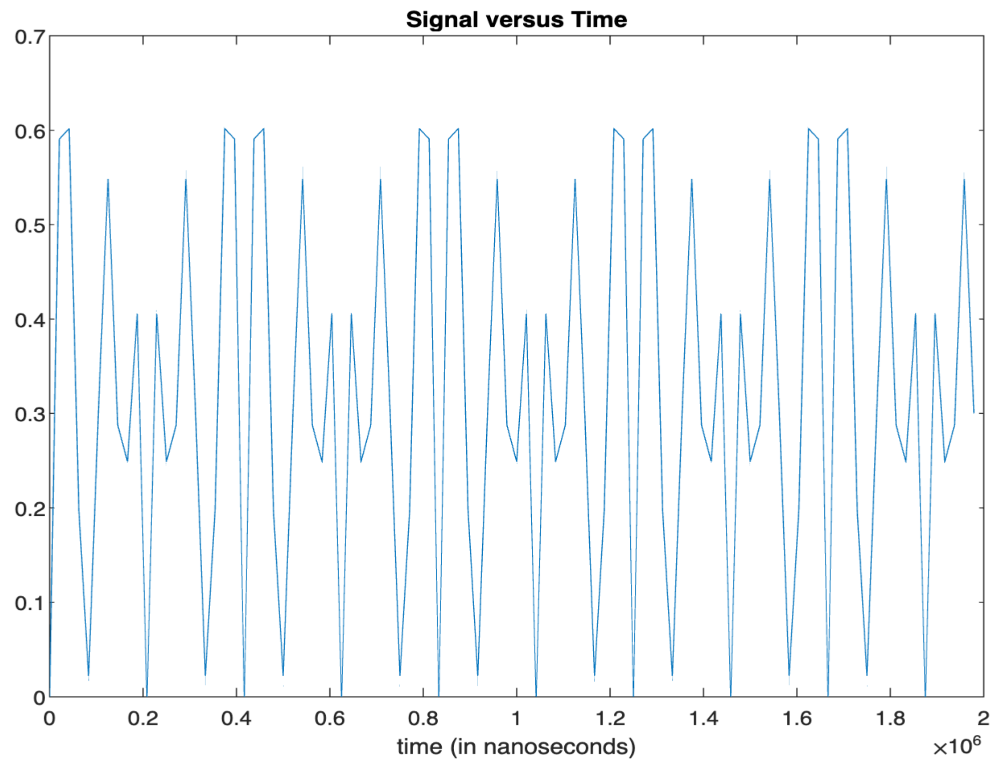


Figure 3: Input Test Signal

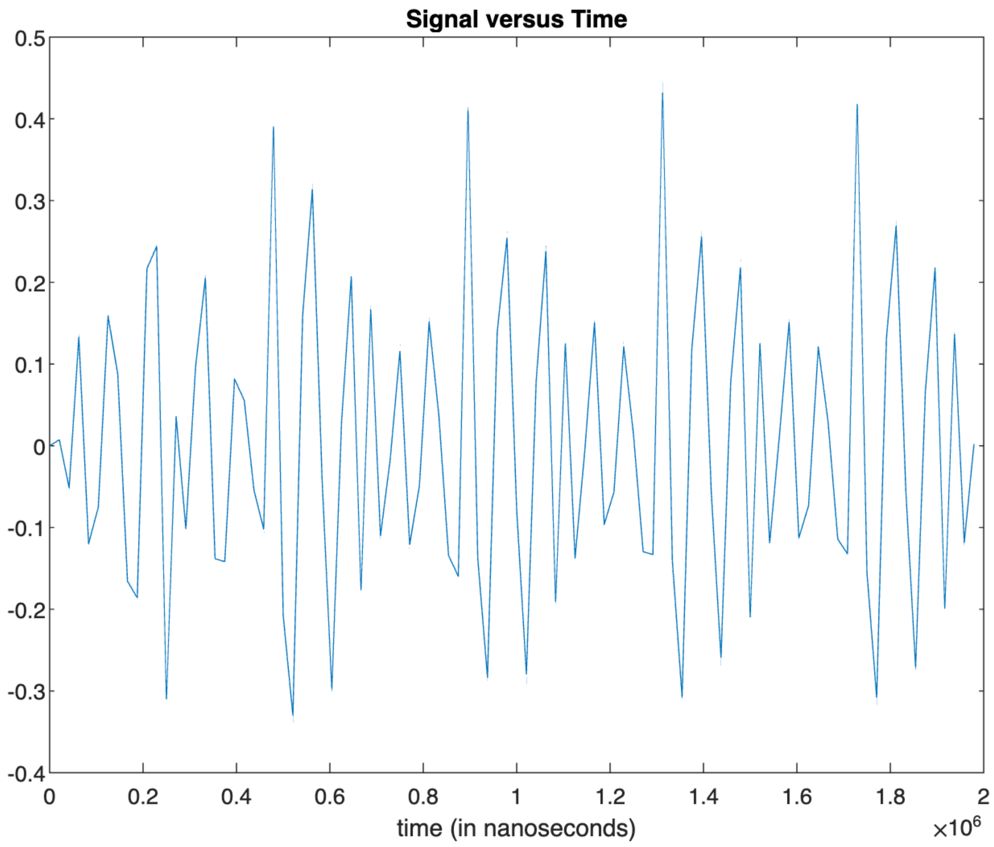


Figure 4: Filtered Output of Test Signal

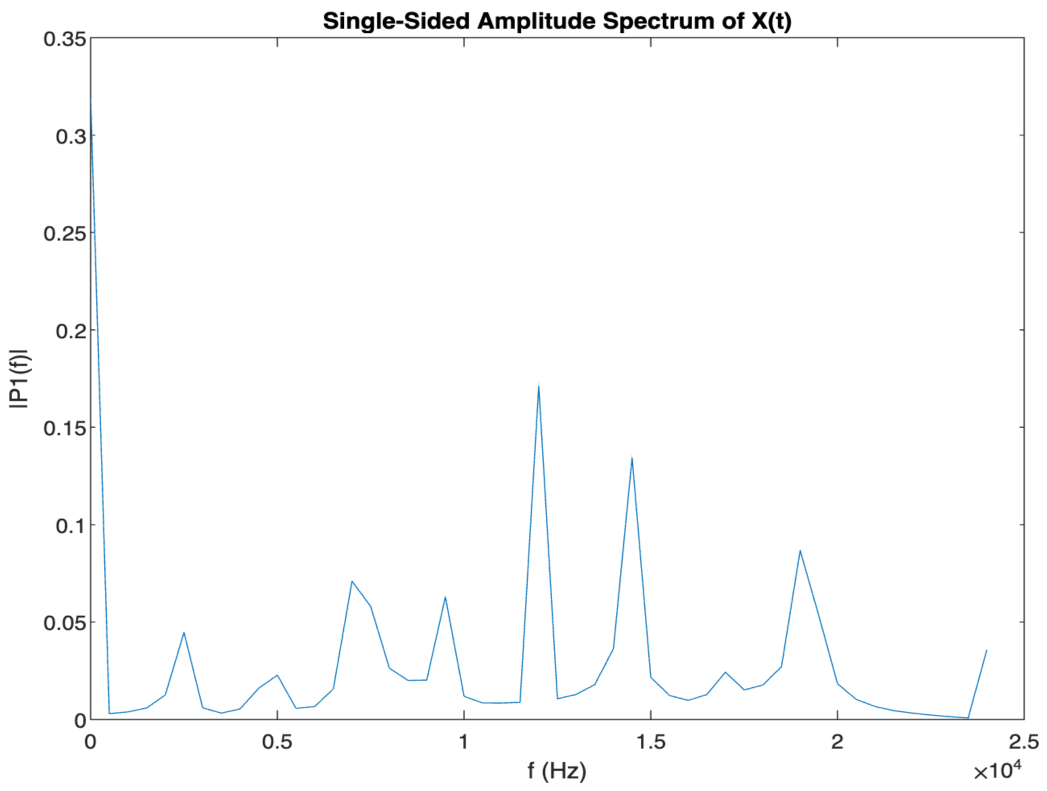


Figure5: FFT of Filter Input Signal

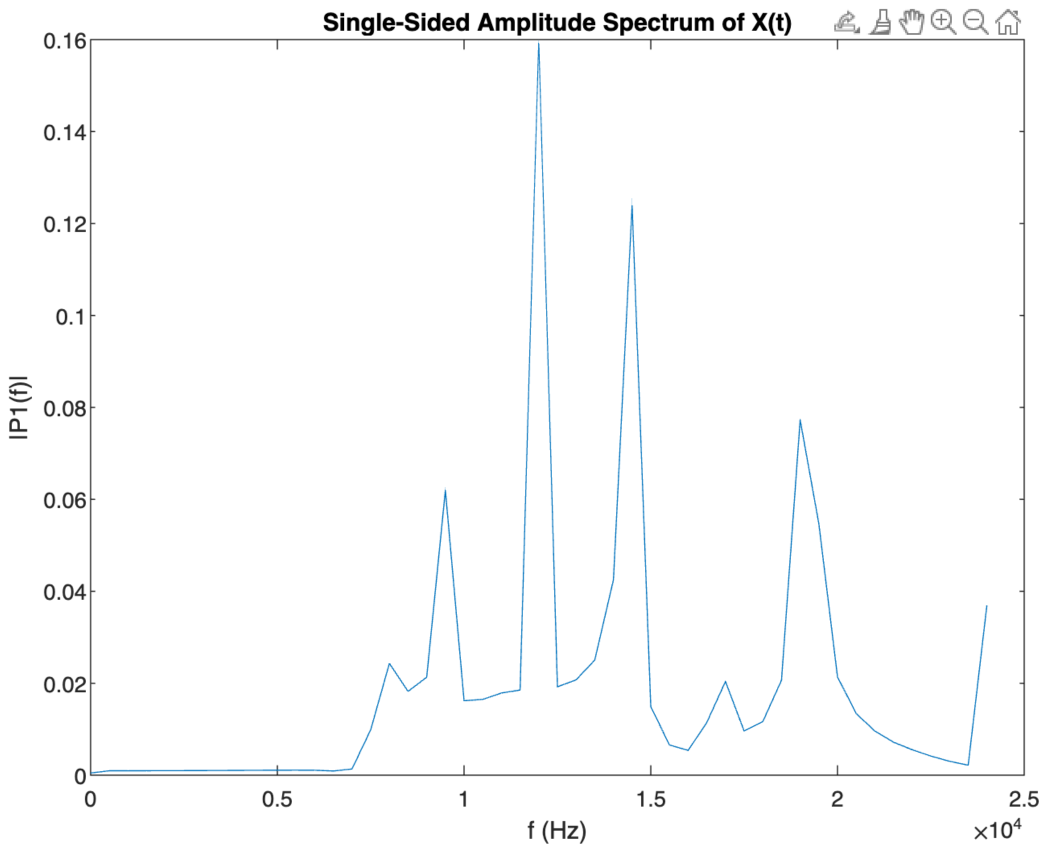


Figure 6: FFT of Filter Output Signal

Verilog Code For IIR:

Filter Module-

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

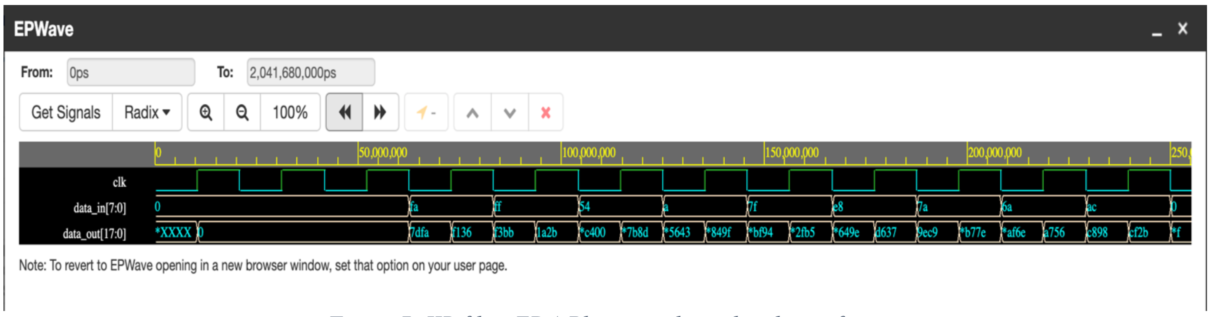
```verilog
module Chevychev_highpass (data_in, data_out, clk, rst);
    output[63:0] data_out;
    input[7:0] data_in;
    input clk, rst;

    parameter b0 = 8'b10000001;
    parameter b1 = 8'b01110110;
    parameter b2 = 8'b10100100;
    parameter b3 = 8'b00101011;
    parameter b4 = 8'b11111111;
    parameter b5 = 8'b00000000;
    parameter b6 = 8'b11010100;
    parameter b7 = 8'b01011011;
    parameter b8 = 8'b10001001;
    parameter b9 = 8'b01111110;

    parameter a1 = 8'b00000000;
    parameter a2 = 8'b11111111;
    parameter a3 = 8'b00010010;
    parameter a4 = 8'b10111110;
    parameter a5 = 8'b01011101;
    parameter a6 = 8'b01110011;
    parameter a7 = 8'b01100100;
    parameter a8 = 8'b01010101;
    parameter a9 = 8'b01010101;

    wire[63:0] data_feedforward;
    wire[63:0] data_feedback;
    reg[17:0] ff_data[1:9];
    reg[17:0] fb_data[1:9];

    assign data_feedforward =
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]+b9*ff_data[9];
    assign data_feedback =
a1*fb_data[1]+a2*fb_data[2]+a3*fb_data[3]+a4*fb_data[4]+a5*fb_data[5]+a6*fb_data[6]
+a7*fb_data[7]+a8*fb_data[8]+a9*fb_data[9];
    assign data_out = (data_feedforward - data_feedback);

    integer k;
    always @ (posedge clk)
        if (rst==1)
        begin
            for (k=1;k<=9;k=k+1)
            begin
                ff_data[k]<=0;
                fb_data[k]<=0;
            end
        end

        else
        begin
            ff_data[1]<=data_in;
            fb_data[1]<=data_out;
            for (k=2;k<=9;k=k+1)
            begin
                ff_data[k]<=ff_data[k-1];
                fb_data[k]<=fb_data[k-1];
            end
        end

endmodule
```

Testbench Module-
The following code module is a testbench module for testing filter modules. 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[63:0] data_out;

    Chevychev_highpass filter1(.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 7.


EPWave

From: 0ps To: 2,041,680,000ps

Note: To revert to EPWave opening in a new browser window, set that option on your user page.
Get Signals
Radix
100%
0
50,000,000
100,000,000
150,000,000
200,000,000
250,
clk
data_in[7:0]
0 fa ff 54 a 7f e8 7a 6a ac 0
data_out[17:0]
*XXXX 0 7dfa f136 f3bb 1a2b *c400 *7b8d *5643 *849f *bf94 *2fb5 *649e d637 9ec9 *b77e *af6e a756 c898 cf2b *f

Figure 7: IIR filter EDA Playground simulated waveform

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-

```
n=9;
rp=0.5;
fs = 48000;
f1 = 8000;
[b,a] = cheby1(n,rp,2*f1/fs,'high');

% 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,a,x);

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

% Deuantizing 18bit digital output to analogue signal
n = 64;
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 = 1;
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 comparison is given below-

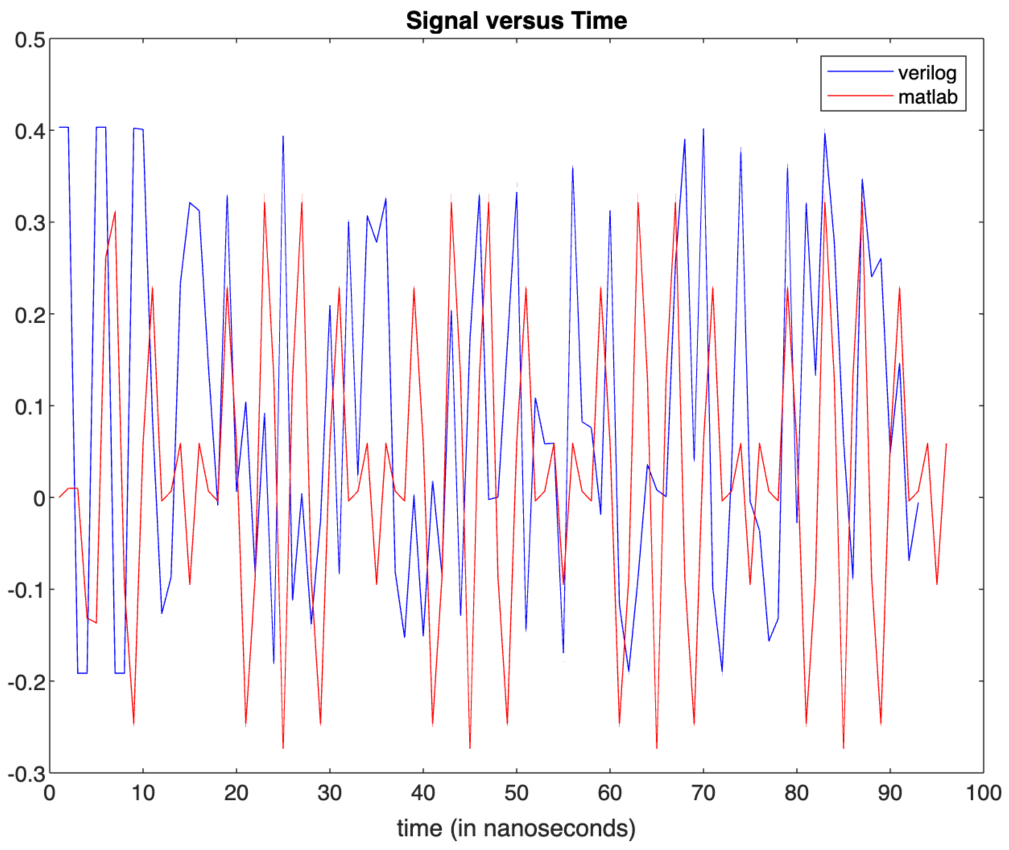


Figure 8: comparison of filter output simulation in time domain

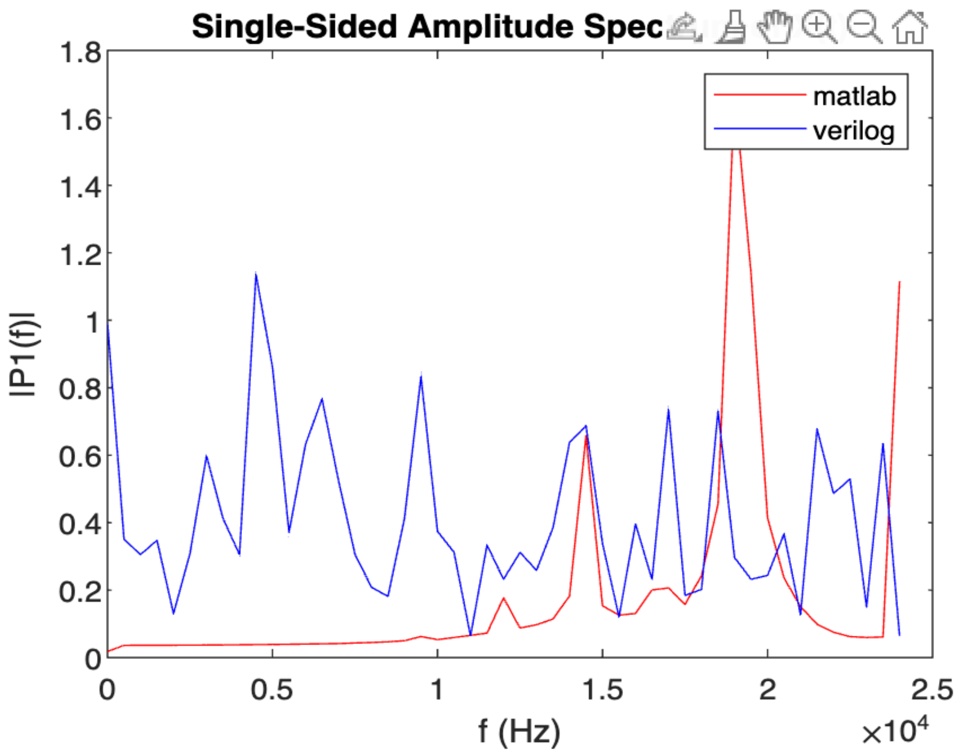


Figure 9: Comparison of filter output simulation in frequency domain in 18-bit dequantization

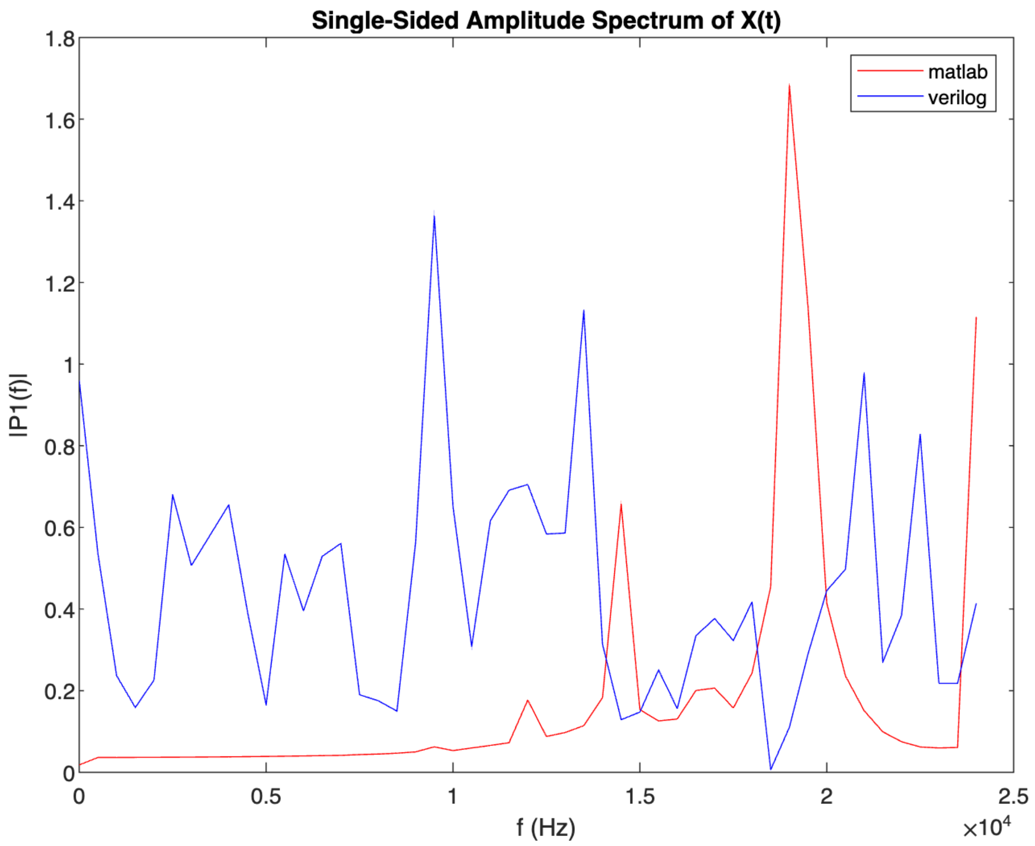


Figure 10: Comparison of filter output simulation in frequency domain in 64-bit dequantization

Synthesize in Cyclone V by Quartus Prime Lite for IIR:

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 – 741 / Total- 32,070 (< 2 %)
Total registers	584
Total pins	74 / 457 (16 %)
Total DSP Blocks	36 / 87 (41 %)

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

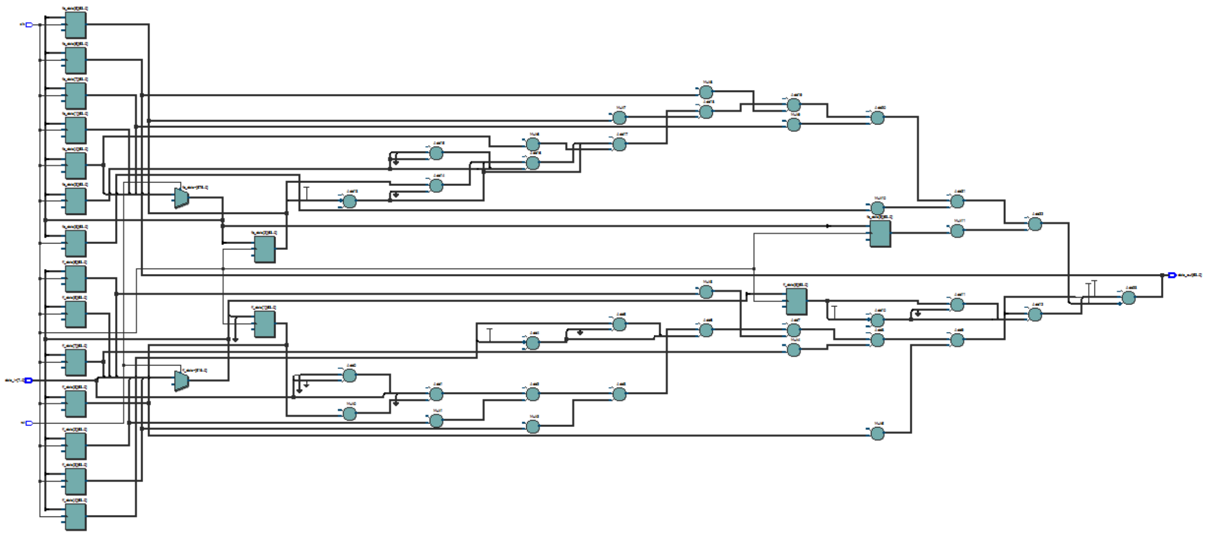


Figure 11: Comparison of filter output simulation in frequency domain in 18-bit dequantization

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 IIR 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.