DDS-FIR-Low-Pass-Filter

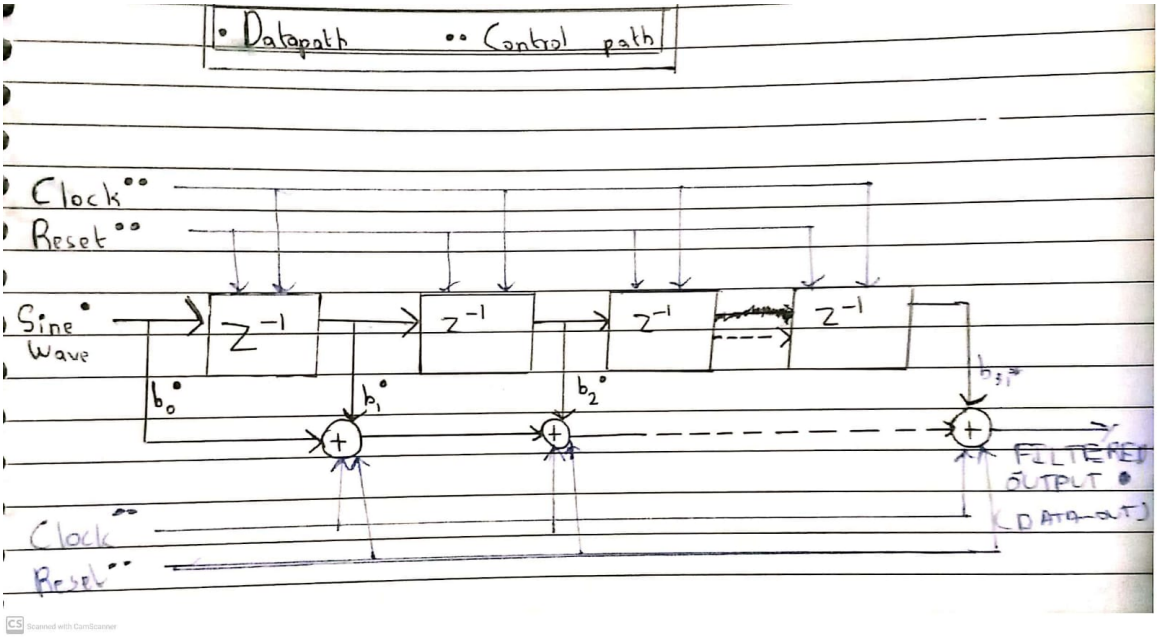
Syed Asghar Abbas Zaidi
s.aazaidi2001@gmail.com

May 26, 2024

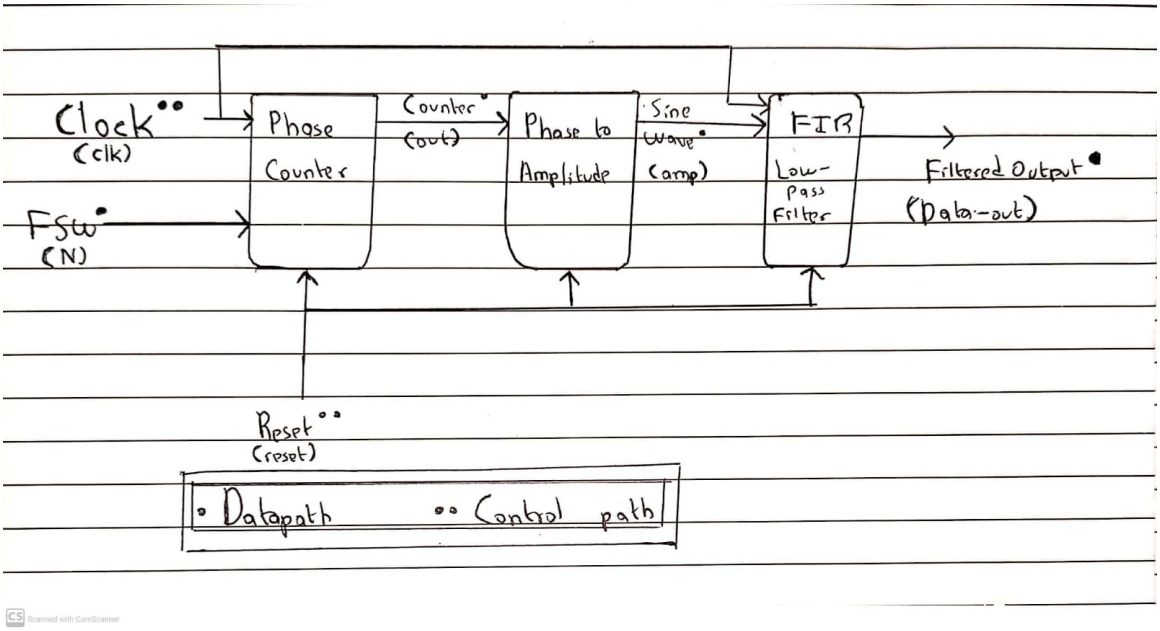
Contents

1 System Architecture

(a) Data Path∗ Control Path∗∗



(b) Fir Low Pass Filter Architecture



(c) Top Level Architecture of Entire System

Figure 1: System Architecture

2 Filter Design In Matlab

I used the Matlab Filter Designer (Previously fdaTool) to design a Low Pass Filter which allows frequencies till 75kHz and stops frequencies greater than 125kHz. It is a 31 Order Filter. Sampling Frequency of the filter is kept 1MHz matching that of Direct Digital Synthesizer.

2.1 Configuring Low-Pass Filter in Matlab Filter Designer

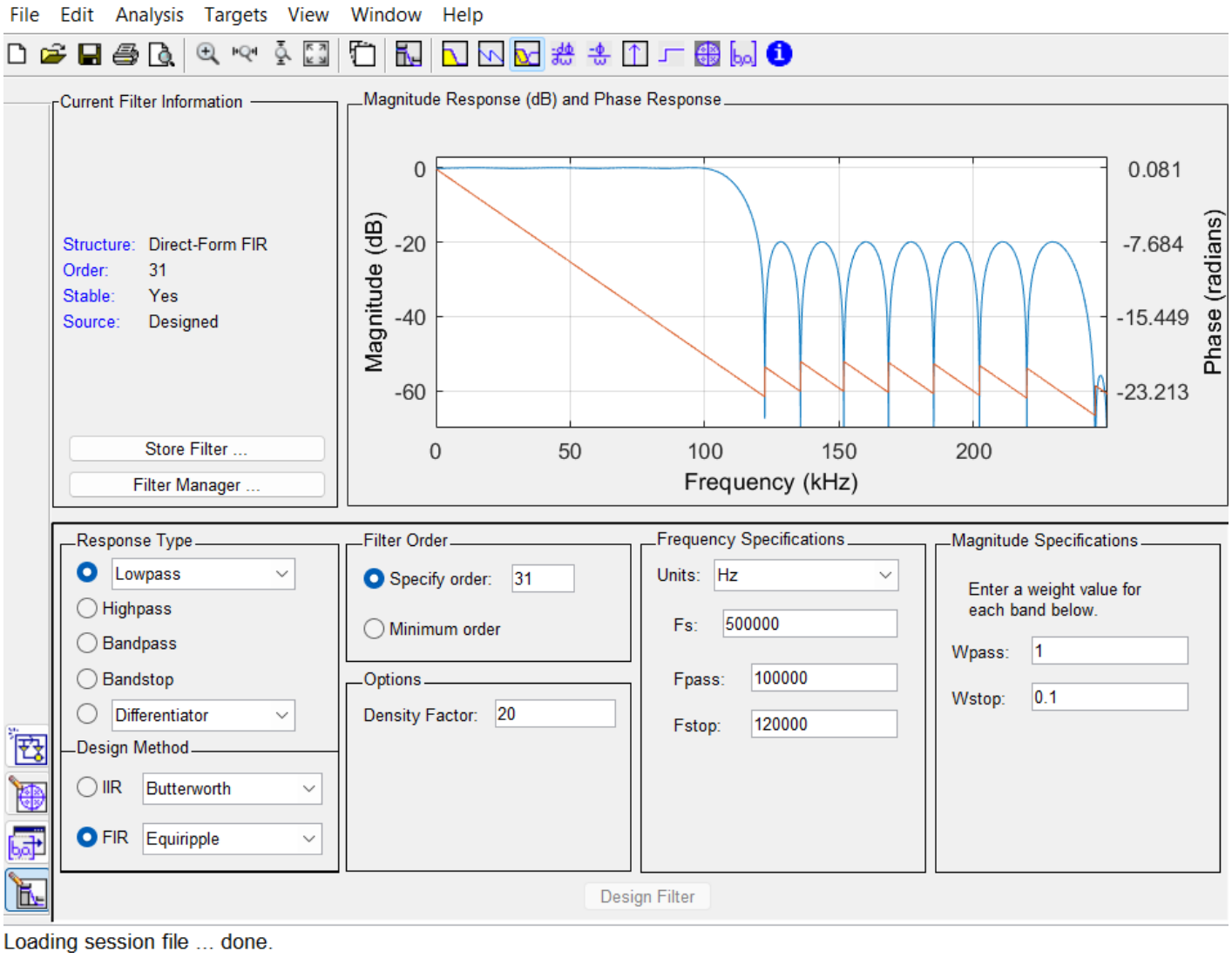


Figure 2: LPF Configuration, Magnitude and Phase Response

I have exported the coefficients of the above filter with the name of 'LPF' in the workspace. I then made all negative coefficients in the LPF matrix equal to zero, in order to not deal with signed binary representation in Verilog.

2.2 Filter after setting negative coefficients to zero

We decided to zero' the co-efficients cause we didn't cater to negative values in our Verilog code. This is the reason why we vertically off-set the Sine Signal of ours by "100" to begin with. As we aren't dealing in negatives, we disregarded those values.

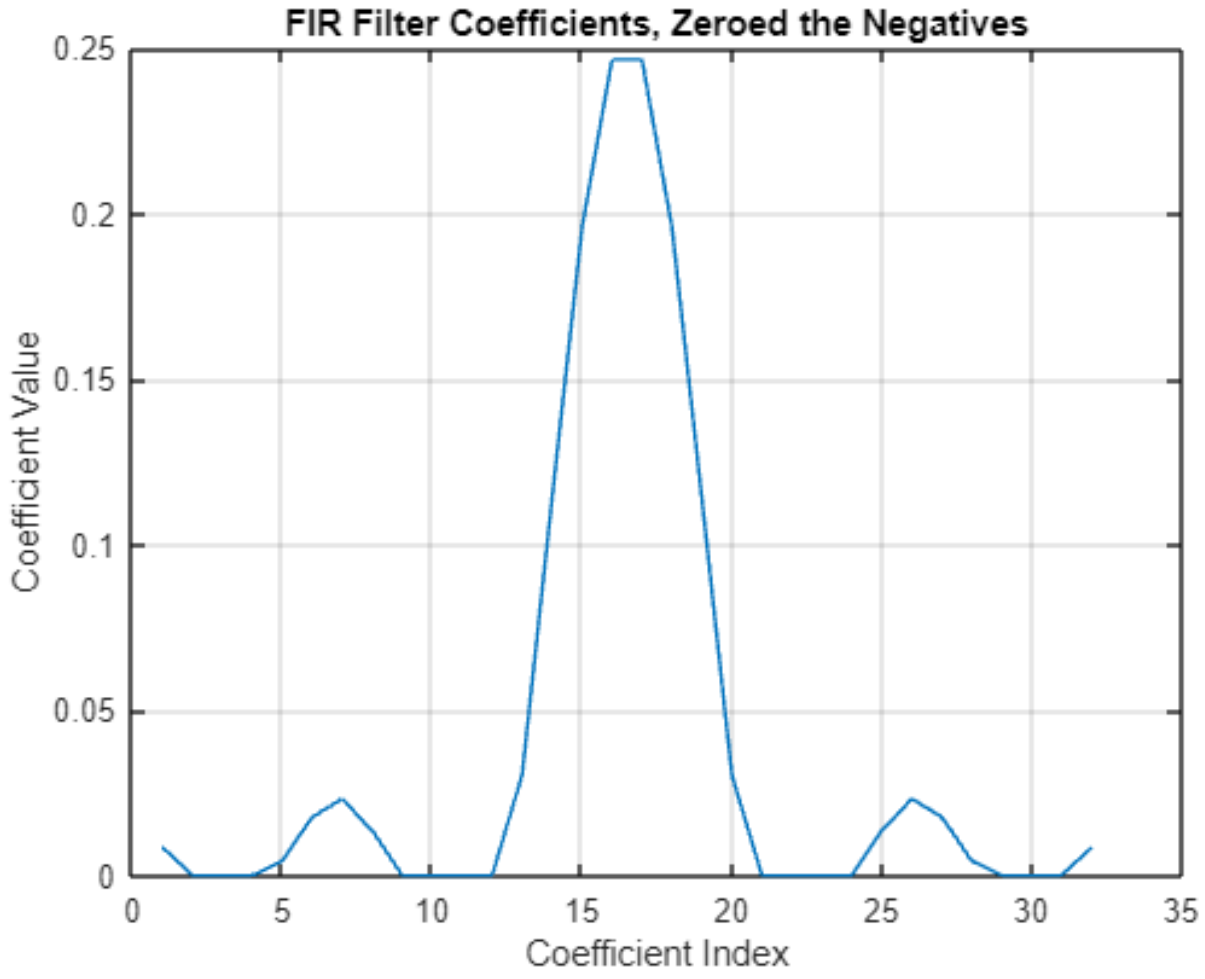
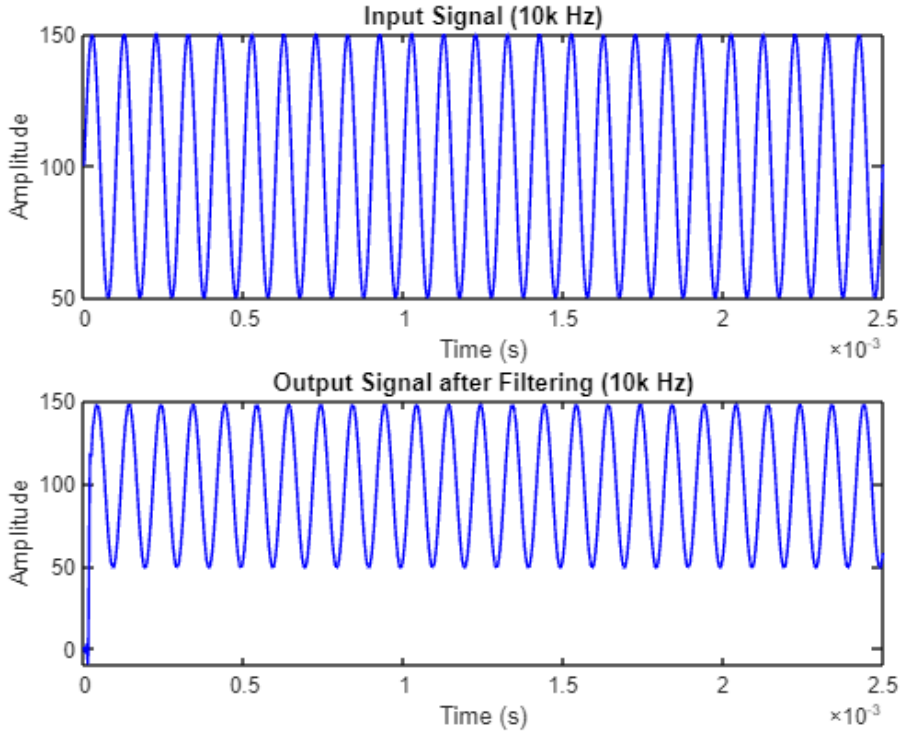


Figure 3: LPF Filter Co-efficients after Zeroing the negatives

2.3 Testing the Filter on Matlab-Generated Sine Waves (IDEAL CASE)

Will be testing for 10kHz and 200kHz frequencies.

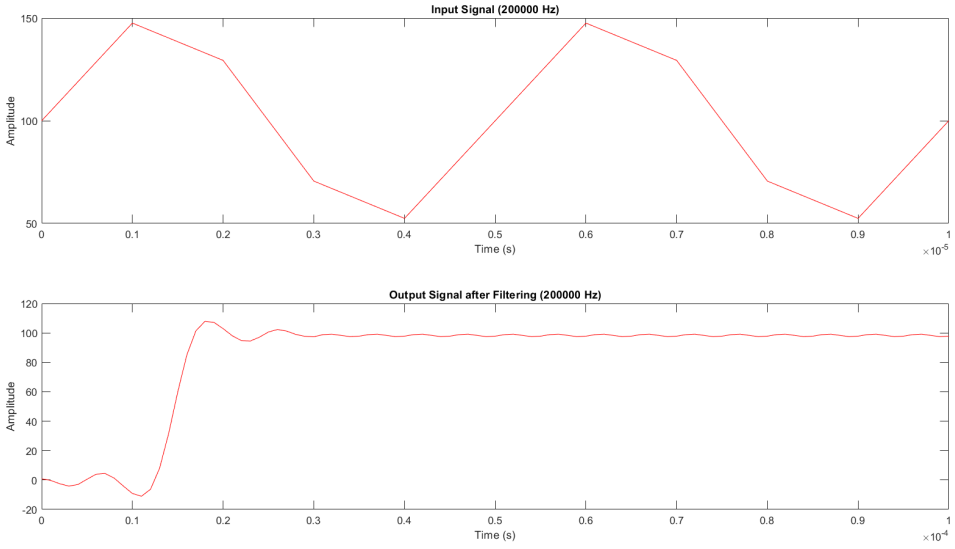
2.3.1 10kHz Matlab Generated Sine Wave Results



(a) 10kHz Sine Wave before and after being filtered

The reason the output signal begins from zero is cause it takes abit of time for the co-efficients to fill up

2.3.2 200kHz Matlab Generated Sine Wave Results



(a) 200kHz Sine Wave before and after being filtered

As can be observed, the high frequency is being attenuated. Please don't mind the rough input signal, due to sampling frequency being 1 MHz, it is resulting in barely 5 samples taking place within the span it takes 200kHz signal to complete its' cycle!

2.4 Converting Filter Coefficients to 8 Bit Binary Representation

Cause we don't have floating point representation in our Matlab, we will be converting individual coefficient into their respective 8-bit binary representation!

```
bits = 8; % Number of bits for the binary representation
coeffs_binary = zeros(size(coefficients), 'uint8'); % Initialize array
   to store binary representations

% Convert each coefficient to binary
for i = 1:numel(coefficients)
    % Scale the coefficient to the range [0, 255] for 8-bit
       representation
    scaled_coeff = round(coefficients(i) * 255);

    % Ensure the scaled coefficient is within the valid range
    scaled_coeff = max(0, min(255, scaled_coeff));

    % Convert the scaled coefficient to binary
    coeffs_binary(i) = uint8(scaled_coeff);
end

% Display the binary representation of coefficients
disp('Binary representation of filter coefficients:');
disp(coeffs_binary);
```

2 0 0 0 1 4 6 3 0 0 0 0 8 29 50 63 63 50 29 8 0 0 0 0 3 6 4 1 0 0 0 2

3 Filter Implementation in Verilog

3.1 FIR Low Pass Filter:

3.1.1 Code:

```
module FIR_Gaussian_Lowpass(Data_out, Data_in, clk, reset);
    parameter order = 32;
    parameter word_size_in = 8;
    parameter word_size_out = 2 * word_size_in + 2;
//      2     0     0     0     1     4     6     3 ..
    parameter b0 = 8'd2;
    parameter b1 = 8'd0;
    parameter b2 = 8'd0;
    parameter b3 = 8'd0;
    parameter b4 = 8'd1;
```

```
    parameter b5 = 8'd4;
    parameter b6 = 8'd6;
    parameter b7 = 8'd3;
//0     0     0     0     8     29     50     63     63
    parameter b8 = 8'd0;
    parameter b9 = 8'd0;
    parameter b10 = 8'd0;
    parameter b11 = 8'd0;
    parameter b12 = 8'd8;
    parameter b13 = 8'd29;
    parameter b14 = 8'd50;
    parameter b15 = 8'd63;
//      63 50    29     8     0     0     0     0
    parameter b16 = 8'd63;
    parameter b17 = 8'd50;
    parameter b18= 8'd29;
    parameter b19 = 8'd8;
    parameter b20 = 8'd0;
    parameter b21 = 8'd0;
    parameter b22 = 8'd0;
    parameter b23 = 8'd0;
 //     3     6     4     1     0     0     0     2
    parameter b24 = 8'd3;
    parameter b25 = 8'd6;
    parameter b26 = 8'd4;
    parameter b27 = 8'd1;
    parameter b28 = 8'd0;
    parameter b29 = 8'd0;
    parameter b30 = 8'd0;
    parameter b31 = 8'd2;

    output [word_size_out - 1:0] Data_out;
    input [word_size_in - 1:0] Data_in;
    input clk;
    input reset;

    reg [word_size_in - 1:0] Samples [1:order];
    integer k;

    assign Data_out =
        b0 * Data_in +
        b1 * Samples[1] +
        b2 * Samples[2] +
        b3 * Samples[3] +
        b4 * Samples[4] +
        b5 * Samples[5] +
        b6 * Samples[6] +
        b7 * Samples[7] +
        b8 * Samples[8] +
        b9 * Samples[9] +
```

```
        b10 * Samples[10] +
        b11 * Samples[11] +
        b12 * Samples[12] +
        b13 * Samples[13] +
        b14 * Samples[14] +
        b15 * Samples[15] +
        b16 * Samples[16] +
        b17 * Samples[17] +
        b18 * Samples[18] +
        b19 * Samples[19] +
        b20 * Samples[20] +
        b21 * Samples[21] +
        b22 * Samples[22] +
        b23 * Samples[23] +
        b24 * Samples[24] +
        b25 * Samples[25] +
        b26 * Samples[26] +
        b27 * Samples[27] +
        b28 * Samples[28] +
        b29 * Samples[29] +
        b30 * Samples[30] +
        b31 * Samples[31];

    always @(posedge clk) begin
        if (reset == 1) begin
            for (k = 1; k <= order; k = k + 1)
                Samples[k] <= 0;
        end else begin
            Samples[1] <= Data_in;
            for (k = 2; k < order; k = k + 1)
                Samples[k] <= Samples[k - 1];
        end
    end
endmodule
```

3.1.2 Schematic:

This schematic illustrates the implementation of the Verilog code for a low pass filter within the context of a Digital Signal Synthesizer (DDS). Within the DDS architecture, the Low-Pass Filter is intricately connected to other essential components to fulfill its role effectively.

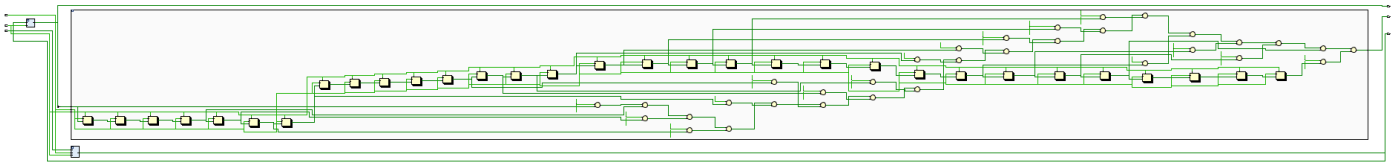


Figure 6: Low Pass Filter Schematic

3.2 Top Level Module:

Showcasing which other components (Phase Counter AND Phase to Amplitude) it is connected to. As I have already shown that in previous assignment, I will be showcasing only the $\text{Top}_{L}evelModulechangesImadeand$

3.2.1 Code:

```
‘timescale 1ns / 1ps

module toplevel(
    input [9:0] N, //FSW
    input reset,
    input clk,
    output wire [9:0] out, //counter
    output wire [9:0] amp,//sine wave

    //word_size_out = 2 * word_size_in + 2,
    output wire [17:0] Data_out
    );

    PhaseAccumulator PA (.N(N), .reset(reset), .clk(clk), .out(out));
    Phase_to_Amplitude_Converter P_to_A (.out(out), .reset(reset), .amp(amp));
    FIR_Gaussian_Lowpass FIR (Data_out, amp, clk, reset);
  endmodule
```

3.2.2 Schematic:

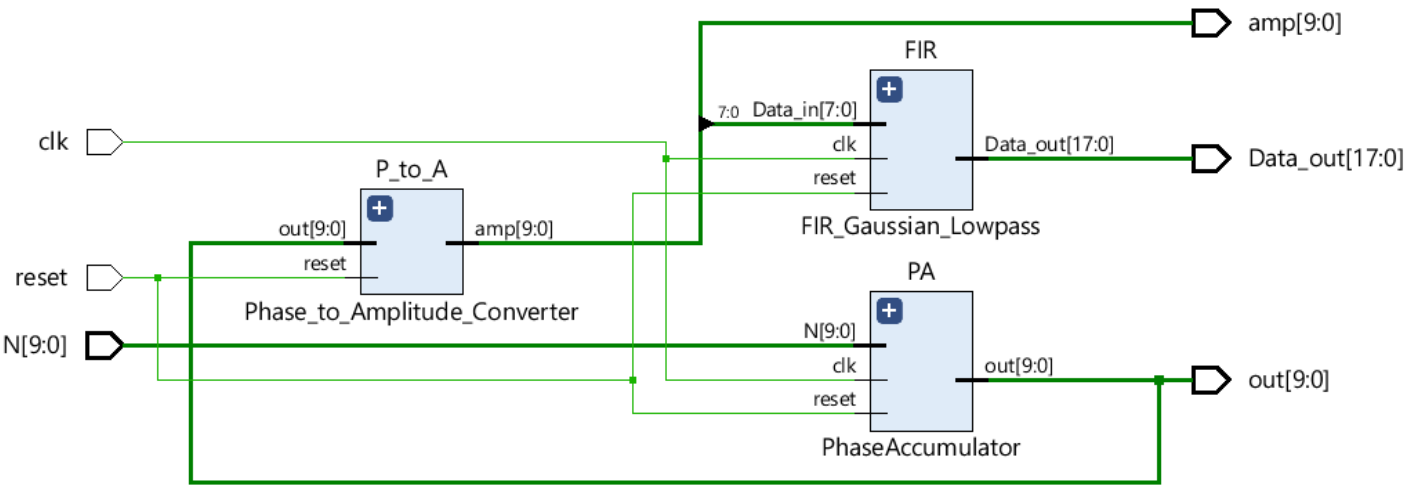


Figure 7: Top Level Schematic

3.3 Test-bench:

```
`timescale 1ns / 1ps
module simulation;
    reg [9:0] N; //fsw
    reg reset;
    reg clk;
    wire [9:0] out;
    wire [9:0] amp;
    wire [17:0] Data_out;

    // Instantiate the module
    toplevel UUT (.N(N), .reset(reset), .clk(clk), .out(out),
    .amp(amp), .Data_out(Data_out) );

    // Initialize signals
    initial begin
        clk = 0;
        $dumpfile("simulation.txt"); // Specify the name of the
        output text file
        N = 205; //fsw
        reset = 1;
        #5 reset = 0;
    end

    initial begin
        clk = 0;
        forever #500 clk = ~clk; //this creates a 1 MHz clock!
    end
    //always #5 clk = ~clk;
    //$display("Simulating output Phase Accumulator");
    //$monitor($time,,, "clk = %b N = %b out = %b ", clk, N, out);
    // Below command is for monitoring the output
    always @(posedge clk) begin
        $display("Time = %t, Counter = %0d, amp = %0d, Data_out = %d",
        $time, UUT.out, UUT.amp, UUT.Data_out);
    end


    initial #1000000 $finish;

endmodule
```

4 Results

In Verilog simulation I printed the Time, Counter, amp (Sine wave amplitude values), and Data_out (filtered output amplitude values) in the logs.
Below I will showcasing the Log results and Simulation from FSW = 205.

4.1 Log Results

I **disregarded** the first 32 values which were used to fill the filter, then copied these output log values into a .txt file and used Matlab to parse through it and generate one wave using AMP output values and other using output values, both against the time values. However for filtered output I also divided the values by 256 and then plotted them.

```
.
.
.
Time =              31500000, Counter = 211, amp = 148, Data_out =  43575
Time =              32500000, Counter = 416, amp = 124, Data_out =  45833
Time =              33500000, Counter = 621, amp = 63, Data_out =  35956
Time =              34500000, Counter = 826, amp = 55, Data_out =  27634
Time =              35500000, Counter = 7, amp = 103, Data_out =  32352
.
.
.
```

4.2 Simulation:

I was getting the following simulation results. Although not completely shown, it takes 32 clock cycles for the Filter to fill up, that is why LPF is giving **X** (Don't Care) cause it's garbage data which isn't useful for us. After that however, we can see the LPF giving us useful data.

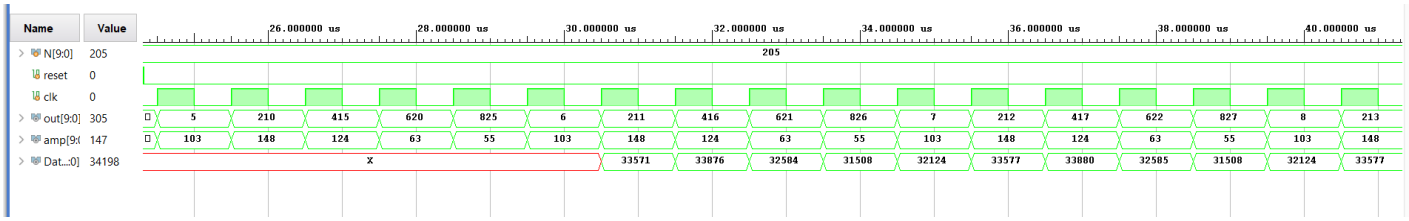


Figure 8: Simulation for 10kHz Sine Wave

4.3 DDS_Output

I used the following Matlab Script for this. I haven't added the code for "plotting" itself cause it's redundant.

4.3.1 Matlab Script Used For Plotting:

```
1 fileID4 = fopen('Low_FIR_Second.txt', 'r');
2 fileID5 = fopen('High_FIR_Second.txt', 'r');
3 %data1 = fscanf(fileID, 'Time= %f, Counter= %d, DDS_Sin= %d\n', [3,
     Inf]);
4 data4 = fscanf(fileID4, 'Time = %f, Counter = %d, amp = %d, Data_out =
     %d\n', [4, Inf]);
```

```
 data5 = fscanf(fileID5, 'Time = %f, Counter = %d, amp = %d, Data_out =
     %d\n', [4, Inf]);
 fclose(fileID4);
 fclose(fileID5);


time4 = data4(1, :);
counter4 = data4(2, :);
amp4 = data4(3, :);
Data_out4 = (data4(4,:)/255); %normalized

time5= data5(1, :);
counter5 = data5(2, :);
amp5 = data5(3, :);
Data_out5 = (data5(4,:)/255); %normalized
.
.
.
% We then just plotted amp4 on time4, and amp5 on time 5
```

4.3.2 Plots:

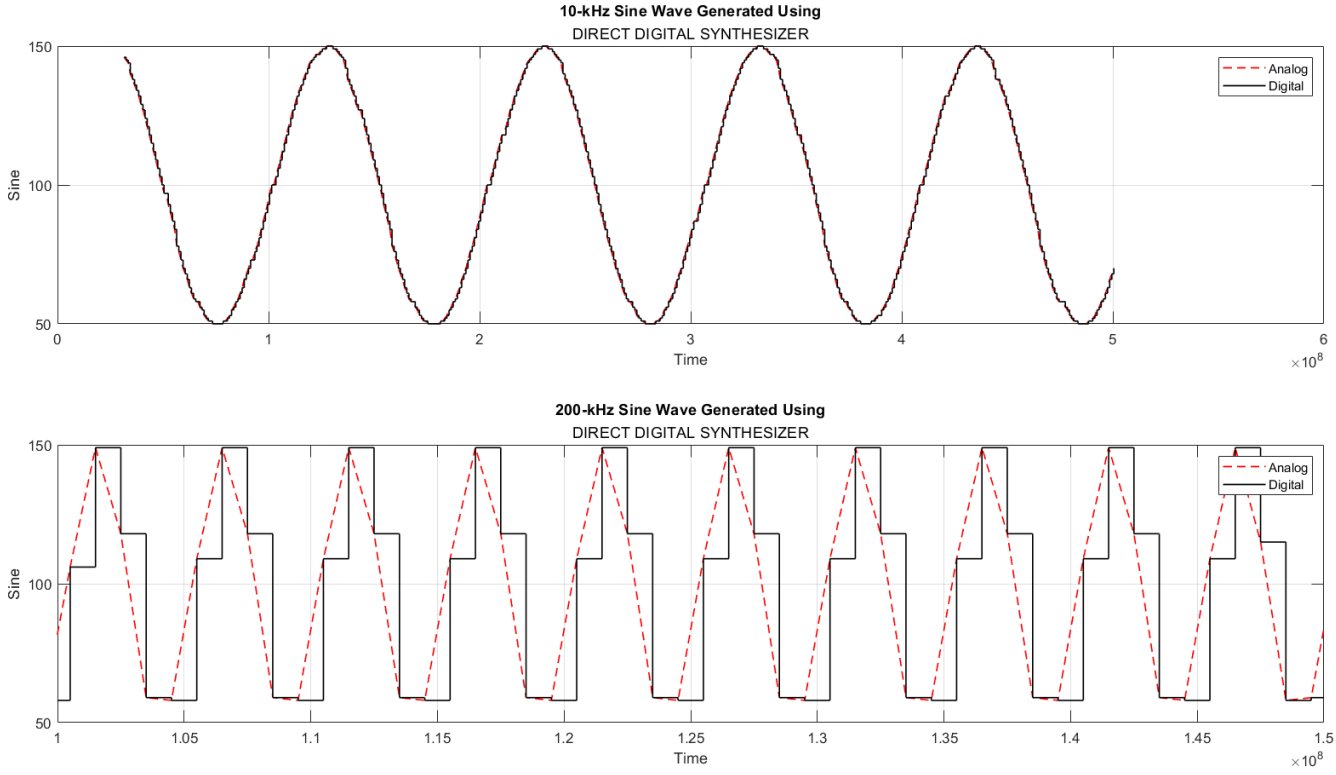


Figure 9: DDS Phase-amp output

4.4 Realized LPF_Output

I just plotted the graph Data_out4 over time4 and Data_out5 over time5. All the codes will be attached in the appendix! As can be observed clearly, the high-frequency signal is getting attenuated.

4.4.1 Plots:

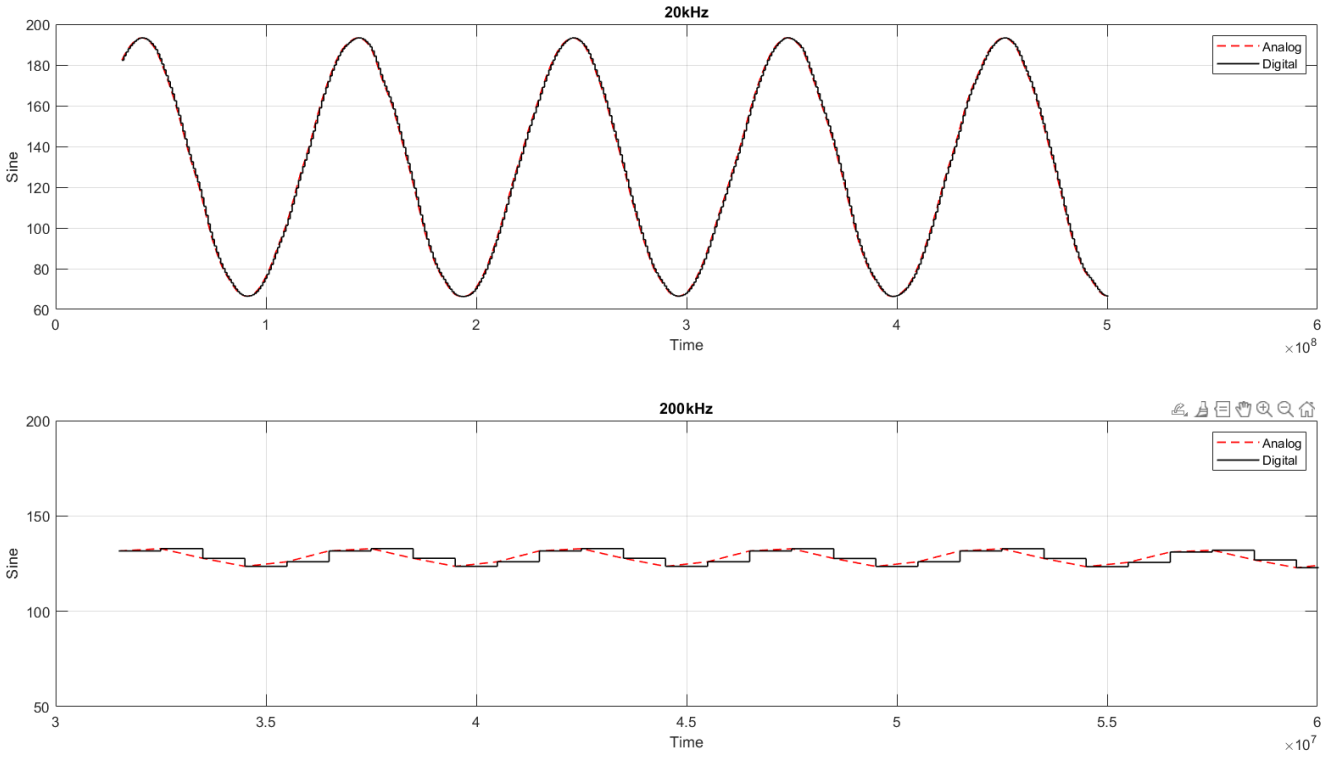


Figure 10: DDS Phase-amp output

5 Conclusion

To conclude, this assignment delved into the design and implementation of a Digital Signal Synthesizer (DDS) system, focusing particularly on the development of a FIR Low Pass Filter (LPF) within the larger architecture.

The system architecture was illustrated, showcasing both the data path and control path components, along with the top-level architecture integrating the LPF within the DDS system. The LPF design was initiated in MATLAB, where a Low Pass Filter with specific frequency requirements was designed using the Filter Designer tool. The coefficients of the designed filter were then exported and processed to fit the requirements of the subsequent Verilog implementation.

In Verilog, the FIR LPF was implemented, utilizing the coefficients obtained from MATLAB. The Verilog code was structured to accommodate these coefficients, enabling the LPF to attenuate higher frequency components effectively while allowing lower frequency components to pass through.

The Verilog implementation was validated against Matlab's ideal plots through simulation, verifying and validating the LPF's functionality within the DDS system in Verilog.

Overall, this assignment provided hands-on experience in designing and implementing digital signal processing components, emphasizing the integration of MATLAB design tools with Verilog hardware description language for efficient system realization.

A Verilog Design Code

```
`timescale 1ns / 1ps

module toplevel(
    input [9:0] N, //FSW
    input reset,
    input clk,
    output wire [9:0] out, //counter
    output wire [9:0] amp,//sine wave

    //word_size_out = 2 * word_size_in + 2,
    output wire [17:0] Data_out
    );

    PhaseAccumulator PA (.N(N), .reset(reset), .clk(clk), .out(out));
    Phase_to_Amplitude_Converter P_to_A (.out(out), .reset(reset), .amp(amp));
    FIR_Gaussian_Lowpass FIR (Data_out, amp, clk, reset);
  endmodule


module PhaseAccumulator(
    N,
    reset,
    clk,
    out
    );

    //input/output
    input [9:0] N;
    input reset;
    input clk;
    output reg [9:0] out;

always @(posedge clk or posedge reset) begin
    if (reset)
        out <= 0;
    else begin
        case ({reset, out >= 1024})
            2'b01: out <= 0;
            2'b10: out <= 0;
            default: if (out < 1024) out <= out + N;
        endcase
    end
end
endmodule
module Phase_to_Amplitude_Converter(
    input [9:0] out,
    input reset,
    output reg [9:0] amp //digital sine
```

```
    );

    always @ (*)
        begin
            if (reset)
                amp = 100;
            else if (out >= 0 && out < 11)
                amp <= 103;
            else if (out >= 11 && out < 21)
                amp <= 106;
            else if (out >= 21 && out < 31)
                amp <= 109;
            else if (out >= 31 && out < 41)
                amp <= 112;
            else if (out >= 41 && out < 52)
                amp <= 115;
            else if (out >= 52 && out < 63)
                amp <= 118;
            else if (out >= 63 && out < 73)
                amp <= 121;
            else if (out >= 73 && out < 83)
                amp <= 124;
            else if (out >= 83 && out < 93)
                amp <= 127;
            else if (out >= 93 && out < 104)
                amp <= 129;
            else if (out >= 104 && out < 115)
                amp <= 132;
            else if (out >= 115 && out < 125)
                amp <= 134;
            else if (out >= 125 && out < 135)
                amp <= 136;
            else if (out >= 135 && out < 145)
                amp <= 138;
            else if (out >= 145 && out < 156)
                amp <= 140;
            else if (out >= 156 && out < 166)
                amp <= 142;
            else if (out >= 166 && out < 176)
                amp <= 144;
            else if (out >= 176 && out < 186)
                amp <= 145;
            else if (out >= 186 && out < 196)
                amp <= 146;
            else if (out >= 196 && out < 207)
                amp <= 147;
            else if (out >= 207 && out < 217)
                amp <= 148;
            else if (out >= 217 && out < 227)
                amp <= 149;
```

```
99              else if (out >= 227 && out < 237)
100                 amp <= 149;
101             else if (out >= 237 && out < 247)
102                 amp <= 150;
103             else if (out >= 247 && out < 258)
104                 amp <= 150;
105             else if (out >= 258 && out < 268)
106                 amp <= 150;
107             else if (out >= 268 && out < 278)
108                 amp <= 149;
109             else if (out >= 278 && out < 288)
110                 amp <= 149;
111             else if (out >= 288 && out < 298)
112                 amp <= 148;
113             else if (out >= 298 && out < 309)
114                 amp <= 147;
115             else if (out >= 309 && out < 319)
116                 amp <= 146;
117             else if (out >= 319 && out < 329)
118                 amp <= 145;
119             else if (out >= 329 && out < 334)
120                 amp <= 144;
121             else if (out >= 334 && out < 339)
122                 amp <= 142;
123             else if (out >= 339 && out < 344)
124                 amp <= 140;
125             else if (out >= 344 && out < 355)
126                 amp <= 138;
127             else if (out >= 355 && out < 365)
128                 amp <= 136;
129             else if (out >= 365 && out < 375)
130                 amp <= 134;
131             else if (out >= 375 && out < 385)
132                 amp <= 132;
133             else if (out >= 385 && out < 395)
134                 amp <= 129;
135             else if (out >= 395 && out < 407)
136                 amp <= 127;
137             else if (out >= 407 && out < 418)
138                 amp <= 124;
139             else if (out >= 418 && out < 428)
140                 amp <= 121;
141             else if (out >= 428 && out < 439)
142                 amp <= 118;
143             else if (out >= 439 && out < 449)
144                 amp <= 115;
145             else if (out >= 449 && out < 460)
146                 amp <= 112;
147             else if (out >= 460 && out < 470)
148                 amp <= 109;
```

```
149             else if (out >= 470 && out < 480)
150                 amp <= 106;
151             else if (out >= 480 && out < 490)
152                 amp <= 103;
153             else if (out >= 490 && out < 500)
154                 amp <= 100;
155             else if (out >= 500 && out < 512)
156                 amp <= 97;
157             else if (out >= 512 && out < 522)
158                 amp <= 93;
159             else if (out >= 522 && out < 532)
160                 amp <= 90;
161             else if (out >= 532 && out < 542)
162                 amp <= 87;
163             else if (out >= 542 && out < 552)
164                 amp <= 84;
165             else if (out >= 552 && out < 553)
166                 amp <= 81;
167             else if (out >= 553 && out < 563)
168                 amp <= 78;
169             else if (out >= 563 && out < 573)
170                 amp <= 76;
171             else if (out >= 573 && out < 583)
172                 amp <= 73;
173             else if (out >= 583 && out < 593)
174                 amp <= 70;
175             else if (out >= 593 && out < 604)
176                 amp <= 68;
177             else if (out >= 604 && out < 614)
178                 amp <= 66;
179             else if (out >= 614 && out < 624)
180                 amp <= 63;
181             else if (out >= 624 && out < 634)
182                 amp <= 61;
183             else if (out >= 634 && out < 644)
184                 amp <= 59;
185             else if (out >= 644 && out < 665)
186                 amp <= 58;
187             else if (out >= 665 && out < 675)
188                 amp <= 56;
189             else if (out >= 675 && out < 685)
190                 amp <= 55;
191             else if (out >= 685 && out < 695)
192                 amp <= 53;
193             else if (out >= 695 && out < 705)
194                 amp <= 52;
195             else if (out >= 705 && out < 716)
196                 amp <= 51;
197             else if (out >= 716 && out < 726)
198                 amp <= 51;
```

```
199             else if (out >= 726 && out < 736)
200                 amp <= 50;
201             else if (out >= 736 && out < 746)
202                 amp <= 50;
203             else if (out >= 746 && out < 756)
204                 amp <= 50;
205             else if (out >= 756 && out < 767)
206                 amp <= 50;
207             else if (out >= 767 && out < 777)
208                 amp <= 50;
209             else if (out >= 777 && out < 787)
210                 amp <= 51;
211             else if (out >= 787 && out < 797)
212                 amp <= 51;
213             else if (out >= 797 && out < 807)
214                 amp <= 52;
215             else if (out >= 807 && out < 819)
216                 amp <= 53;
217             else if (out >= 819 && out < 829)
218                 amp <= 55;
219             else if (out >= 829 && out < 839)
220                 amp <= 56;
221             else if (out >= 839 && out < 849)
222                 amp <= 58;
223             else if (out >= 849 && out < 859)
224                 amp <= 59;
225             else if (out >= 859 && out < 870)
226                 amp <= 61;
227             else if (out >= 870 && out < 880)
228                 amp <= 63;
229             else if (out >= 880 && out < 890)
230                 amp <= 66;
231             else if (out >= 890 && out < 900)
232                 amp <= 68;
233             else if (out >= 900 && out < 910)
234                 amp <= 70;
235             else if (out >= 910 && out < 921)
236                 amp <= 73;
237             else if (out >= 921 && out < 932)
238                 amp <= 76;
239             else if (out >= 932 && out < 942)
240                 amp <= 78;
241             else if (out >= 942 && out < 952)
242                 amp <= 81;
243             else if (out >= 952 && out < 962)
244                 amp <= 84;
245             else if (out >= 962 && out < 973)
246                 amp <= 87;
247             else if (out >= 973 && out < 983)
248                 amp <= 90;
```

```
            else if (out >= 983 && out < 993)
                amp <= 93;
            else if (out >= 993 && out < 1003)
                amp <= 97;
            else if (out >= 1003 && out < 1013)
                amp <= 100;
            else if (out >= 1013 && out < 1023)
                amp <= 100;
            else
                amp <= 100;

    end
endmodule
module FIR_Gaussian_Lowpass(Data_out, Data_in, clk, reset);
    parameter order = 32;
    parameter word_size_in = 8;
    parameter word_size_out = 2 * word_size_in + 2;
//     2     0     0     0     1     4     6     3 ..
    parameter b0 = 8'd2;
    parameter b1 = 8'd0;
    parameter b2 = 8'd0;
    parameter b3 = 8'd0;
    parameter b4 = 8'd1;
    parameter b5 = 8'd4;
    parameter b6 = 8'd6;
    parameter b7 = 8'd3;
//0     0     0     0     8    29    50    63    63
    parameter b8 = 8'd0;
    parameter b9 = 8'd0;
    parameter b10 = 8'd0;
    parameter b11 = 8'd0;
    parameter b12 = 8'd8;
    parameter b13 = 8'd29;
    parameter b14 = 8'd50;
    parameter b15 = 8'd63;
//      63 50    29     8     0     0     0     0
    parameter b16 = 8'd63;
    parameter b17 = 8'd50;
    parameter b18= 8'd29;
    parameter b19 = 8'd8;
    parameter b20 = 8'd0;
    parameter b21 = 8'd0;
    parameter b22 = 8'd0;
    parameter b23 = 8'd0;
 //     3     6     4     1     0     0     0     2
    parameter b24 = 8'd3;
    parameter b25 = 8'd6;
    parameter b26 = 8'd4;
    parameter b27 = 8'd1;
    parameter b28 = 8'd0;
```

```
    parameter b29 = 8'd0;
    parameter b30 = 8'd0;
    parameter b31 = 8'd2;

    output [word_size_out - 1:0] Data_out;
    input [word_size_in - 1:0] Data_in;
    input clk;
    input reset;

    reg [word_size_in - 1:0] Samples [1:order];
    integer k;

    assign Data_out =
        b0 * Data_in +
        b1 * Samples[1] +
        b2 * Samples[2] +
        b3 * Samples[3] +
        b4 * Samples[4] +
        b5 * Samples[5] +
        b6 * Samples[6] +
        b7 * Samples[7] +
        b8 * Samples[8] +
        b9 * Samples[9] +
        b10 * Samples[10] +
        b11 * Samples[11] +
        b12 * Samples[12] +
        b13 * Samples[13] +
        b14 * Samples[14] +
        b15 * Samples[15] +
        b16 * Samples[16] +
        b17 * Samples[17] +
        b18 * Samples[18] +
        b19 * Samples[19] +
        b20 * Samples[20] +
        b21 * Samples[21] +
        b22 * Samples[22] +
        b23 * Samples[23] +
        b24 * Samples[24] +
        b25 * Samples[25] +
        b26 * Samples[26] +
        b27 * Samples[27] +
        b28 * Samples[28] +
        b29 * Samples[29] +
        b30 * Samples[30] +
        b31 * Samples[31];

    always @(posedge clk) begin
        if (reset == 1) begin
            for (k = 1; k <= order; k = k + 1)
                Samples[k] <= 0;
```

```
        end else begin
            Samples[1] <= Data_in;
            for (k = 2; k < order; k = k + 1)
                Samples[k] <= Samples[k - 1];
        end
    end
endmodule
```

B Testbench

Refer to Testbench under Results for code in Section 3.3

C Matlab Code I used for this project

```
%  %Opening the files.
%  fileID = fopen('10kHz.txt', 'r');
%  fileID2 = fopen('125kHz.txt', 'r');
%  fileID3 = fopen('250kHz.txt', 'r');
%  %Reading the data stored in them!
%  data1 = fscanf(fileID, '# KERNEL: Time= %f, Counter= %d, DDS_Sin= %d', [3, Inf]);
%  data2 = fscanf(fileID2, '# KERNEL: Time= %f, Counter= %d, DDS_Sin= %d\n', [3, Inf]);
%  data3 = fscanf(fileID3, '# KERNEL: Time= %f, Counter= %d, DDS_Sin= %d\n', [3, Inf]);
%  % Closing all the files
%  fclose(fileID);fclose(fileID2);fclose(fileID3);
%
%  % Extracting the N (counter), amp and the subsequent time values!
% time1 = data1(1, :);
% counter1 = data1(2, :);
% DDS_SIN1 = data1(3, :);
%
% time2 = data2(1, :);
% counter2 = data2(2, :);
% DDS2 = data2(3, :);
%
% time3 = data3(1, :);
% counter3 = data3(2, :);
% DDS3 = data3(3, :);
%
% % Plot the graphs
% figure;
% subplot(3, 1, 1);
```

```
% plot(time1, DDS_SIN1, '--r', 'LineWidth', 1, 'Color', [1, 0, 0, 0.5])
    ;
% hold on;
% stairs(time1, DDS_SIN1, 'k', 'LineWidth', 1);
% hold off;
% legend('Analog', 'Digital');
% xlabel('Time');
% ylabel('Sine');
% title('10 kHz Sine Wave Generated Using', 'DIRECT DIGITAL SYNTHESIZER
    ');
% grid on;
%
% subplot(3, 1, 2);
% plot(time2, DDS2, '--r', 'LineWidth', 1, 'Color', [1, 0, 0, 0.5]);
% hold on;
% stairs(time2, DDS2, 'k', 'LineWidth', 1);
% hold off;
% legend('Analog', 'Digital');
% xlabel('Time');
% ylabel('Sine');
% title('125-kHz Sine Wave Generated');
% grid on;
%
% subplot(3, 1, 3);
% plot(time3, DDS3, '--r', 'LineWidth', 1, 'Color', [1, 0, 0, 0.5]);
% hold on;
% stairs(time3, DDS3, 'k', 'LineWidth', 1);
% hold off;
% legend('Analog', 'Digital');
% xlabel('Time');
% ylabel('Sine');
% title('250 kHz Sine Wave Generated');
% grid on;


load("FilterCoefficients.mat","DSD2");
coefficients = DSD2;
plot(coefficients);
title('FIR Filter Coefficients');
xlabel('Coefficient Index');
ylabel('Coefficient Value');
grid on;

% Time vector
Fs = 1000000; % Sampling frequency (Hz)
%t = 0:1/Fs:1-1/Fs; % Time vector for 1 second
t = linspace(0,1,Fs);
```

```
73
74 % Generate input signals
75 f1 = 10000; % Frequency of the first sine signal (Hz)
76 f2 = 200000; % Frequency of the second sine signal (Hz)
77 x1 = 50*sin(2*pi*f1*t) + 100; % 10k Hz sine wave
78 x2 = 50*sin(2*pi*f2*t) + 100; % 200000 Hz sine wave
79
80 % Apply the FIR filter to the input signals
81 y1 = filter(coefficients, 1, x1);
82 y2 = filter(coefficients, 1, x2);
83
84 % Plot the input and output signals for 100 Hz
85 figure;
86 subplot(2,1,1);
87 plot(t, x1, 'b');
88 title('Input Signal (10k Hz)');
89 xlabel('Time (s)');
90 ylabel('Amplitude');
91 xlim([0 0.0025]);
92
93
94 subplot(2,1,2);
95 plot(t, y1, 'b');
96 title('Output Signal after Filtering (10k Hz)');
97 xlabel('Time (s)');
98 ylabel('Amplitude');
99 xlim([0 0.0025]);
100
101 % Plot the input and output signals for 200000 Hz
102 figure;
103 subplot(2,1,1);
104 plot(t, x2, 'r');
105 title('Input Signal (200000 Hz)');
106 xlabel('Time (s)');
107 ylabel('Amplitude');
108 xlim([0 0.00001]);
109
110 subplot(2,1,2);
111 plot(t, y2, 'r');
112 title('Output Signal after Filtering (200000 Hz)');
113 xlabel('Time (s)');
114 ylabel('Amplitude');
115 xlim([0 0.0001]);
116
117 clf;
118
119 % Set negative coefficients to zero
```

```
c = coefficients;
coefficients(coefficients < 0) = 0;

plot(coefficients);
title('FIR Filter Coefficients, Zeroed the Negatives');
xlabel('Coefficient Index');
ylabel('Coefficient Value');
grid on;

% Assuming c is the original filter coefficients and coefficients is the modified coefficients

% Visualize original filter magnitude response
[H_original, f] = freqz(c, 1, 1024, 'whole', 44100);
plot(f, abs(H_original));
title('Original Filter Magnitude Response');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
hold on;

% Visualize modified filter magnitude response
[H_modified, ~] = freqz(coefficients, 1, 1024, 'whole', 44100);
plot(f, abs(H_modified));
legend('Original', 'Modified');

fftx2 = fftshift(fft(x2));
figure;
% Determine frequency axis
N = length(x2); % Length of the signal
df = 500000 / N; % Frequency resolution
freq2 = (-Fs/2) + df*(0:N-1); % Frequency axis centered at 0

plot(freq2, fftx2);


% coefficients = DSD2;
% coefficients(coefficients < 0) = 0;
% verilog_coefficients = round(coefficients*255);
% disp(verilog_coefficients);
% Convert filter coefficients to 8-bit binary representation

bits = 8; % Number of bits for the binary representation
coeffs_binary = zeros(size(coefficients), 'uint8'); % Initialize array to store binary representations

% Convert each coefficient to binary
for i = 1:numel(coefficients)
```

```
    % Scale the coefficient to the range [0, 255] for 8-bit
       representation
    scaled_coeff = round(coefficients(i) * 255);

    % Ensure the scaled coefficient is within the valid range
    scaled_coeff = max(0, min(255, scaled_coeff));

    % Convert the scaled coefficient to binary
    coeffs_binary(i) = uint8(scaled_coeff);
end

% Display the binary representation of coefficients
disp('Binary representation of filter coefficients:');
disp(coeffs_binary);

 fileID4 = fopen('Low_FIR_Second.txt', 'r');
 fileID5 = fopen('High_FIR_Second.txt', 'r');
 %data1 = fscanf(fileID, 'Time= %f, Counter= %d, DDS_Sin= %d\n', [3,
    Inf]);
 data4 = fscanf(fileID4, 'Time = %f, Counter = %d, amp = %d, Data_out =
     %d\n', [4, Inf]);
 data5 = fscanf(fileID5, 'Time = %f, Counter = %d, amp = %d, Data_out =
     %d\n', [4, Inf]);
 fclose(fileID4);
 fclose(fileID5);


time4 = data4(1, :);
counter4 = data4(2, :);
amp4 = data4(3, :);
Data_out4 = (data4(4,:)/255); %normalized

time5= data5(1, :);
counter5 = data5(2, :);
amp5 = data5(3, :);
Data_out5 = (data5(4,:)/255); %normalized

% Plot the graphs
figure;
subplot(2, 1, 1);
plot(time4, amp4, '--r', 'LineWidth', 1, 'Color', [1, 0, 0, 0.5]);
hold on;
stairs(time4, amp4, 'k', 'LineWidth', 1);
hold off;
legend('Analog', 'Digital');
xlabel('Time');
ylabel('Sine');
```

```
208 title('10-kHz Sine Wave Generated Using', 'DIRECT DIGITAL SYNTHESIZER')
    ;
209 grid on;
210
211 subplot(2, 1, 2);
212 plot(time5, amp5, '--r', 'LineWidth', 1, 'Color', [1, 0, 0, 0.5]);
213 hold on;
214 stairs(time5, amp5, 'k', 'LineWidth', 1);
215 hold off;
216 legend('Analog', 'Digital');
217 xlabel('Time');
218 ylabel('Sine');
219 title('200-kHz Sine Wave Generated Using', 'DIRECT DIGITAL SYNTHESIZER'
    );
220 xlim([100000000  150000000]);
221 grid on;
222
223
224 % Plot the graphs
225 figure;
226 subplot(2, 1, 1);
227 plot(time4, Data_out4, '--r', 'LineWidth', 1, 'Color', [1, 0, 0, 0.5]);
228 hold on;
229 stairs(time4, Data_out4, 'k', 'LineWidth', 1);
230 hold off;
231 legend('Analog', 'Digital');
232 xlabel('Time');
233 ylabel('Sine');
234 title('20kHz');
235 grid on;
236
237 subplot(2, 1, 2);
238 plot(time5, Data_out5, '--r', 'LineWidth', 1, 'Color', [1, 0, 0, 0.5]);
239 hold on;
240 stairs(time5, Data_out5, 'k', 'LineWidth', 1);
241 hold off;
242 legend('Analog', 'Digital');
243 xlabel('Time');
244 ylabel('Sine');
245 title('200kHz');
246 ylim([50 200]);
247 xlim([30000000 60000000]);
248 grid on;
```