# CE-325: Digital System Design

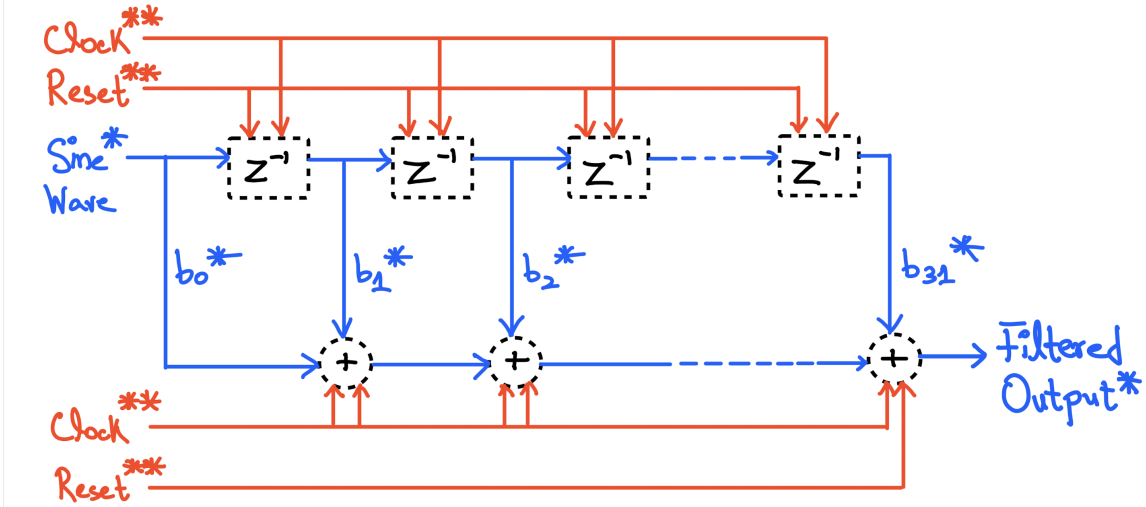
Homework 02

Huzaifah Tariq Ahmed - ha07151

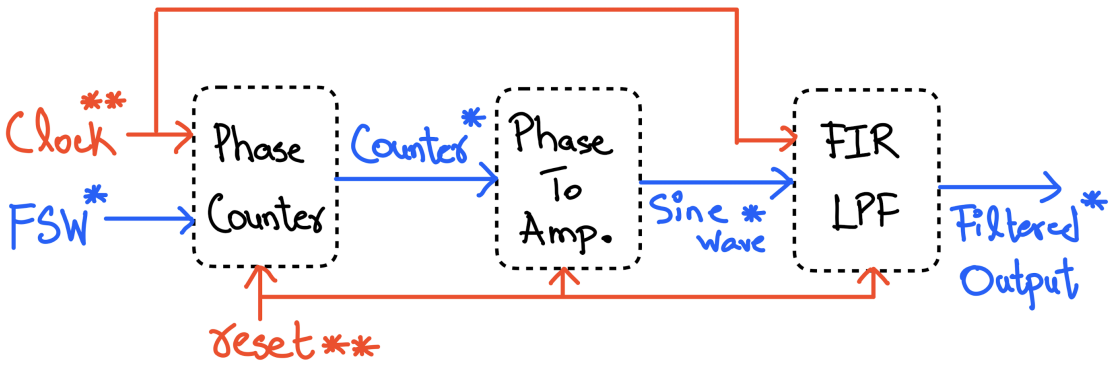
25th March 2024

## 1 System Architecture

Data Path∗ Control Path∗∗




(a) Fir Low Pass Filter Architecture




(b) Top Level Architecture of Entire System

Figure 1: System Architecture

# 2 Filter Design In Matlab

I used the Matlab Filter Designer (Previously FDA Tool) 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 DDS.

## 2.1 Configuring LPF in Matlab Filter Designer

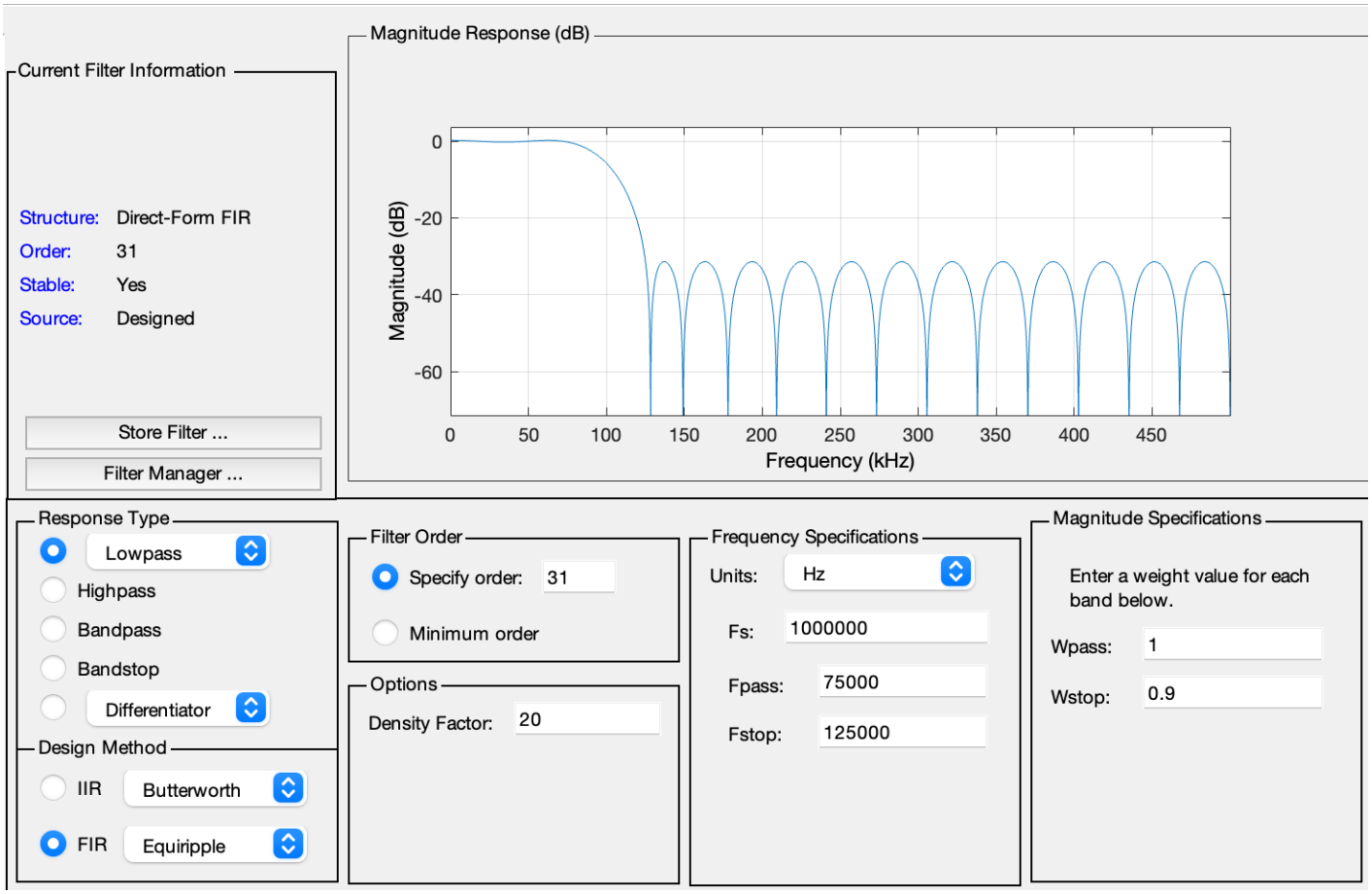



Figure 2: LPF Configuration and Magnitude Response

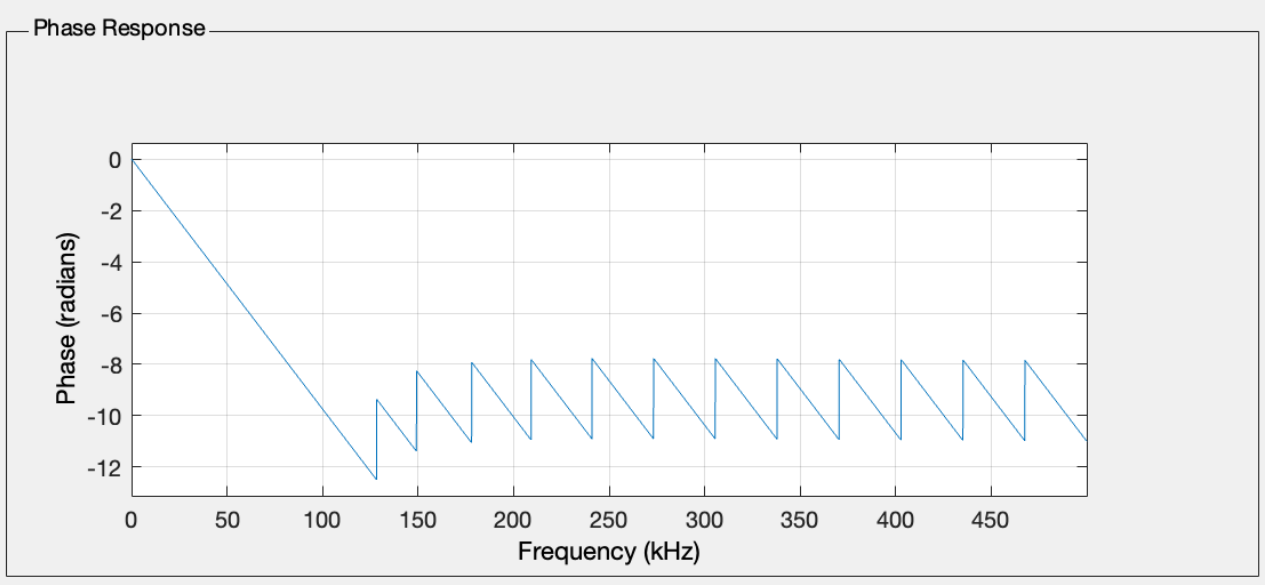



Figure 3: Phase Response of LPF

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

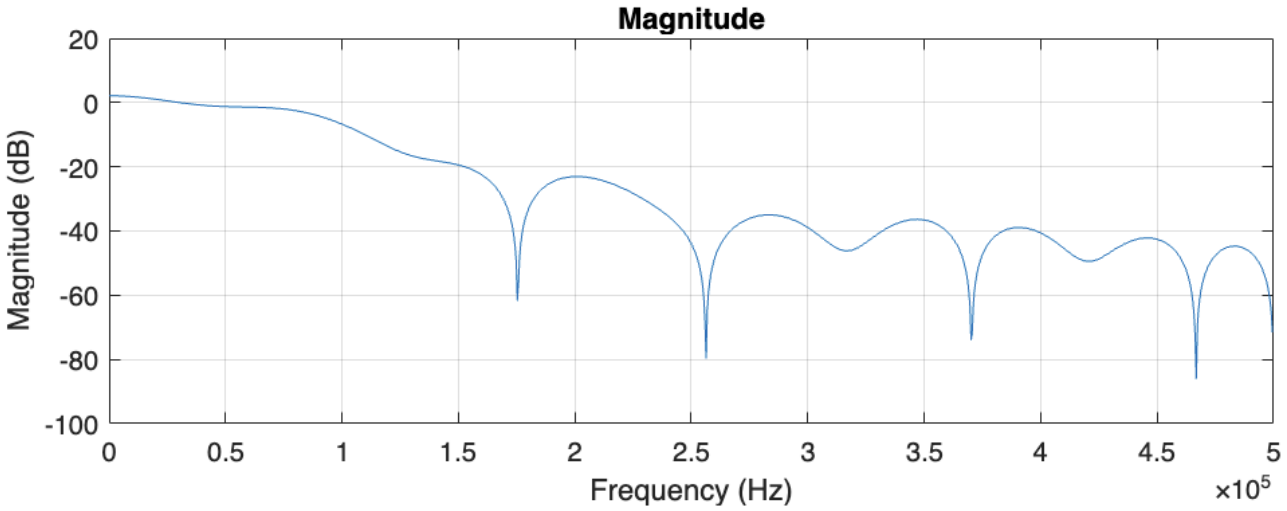


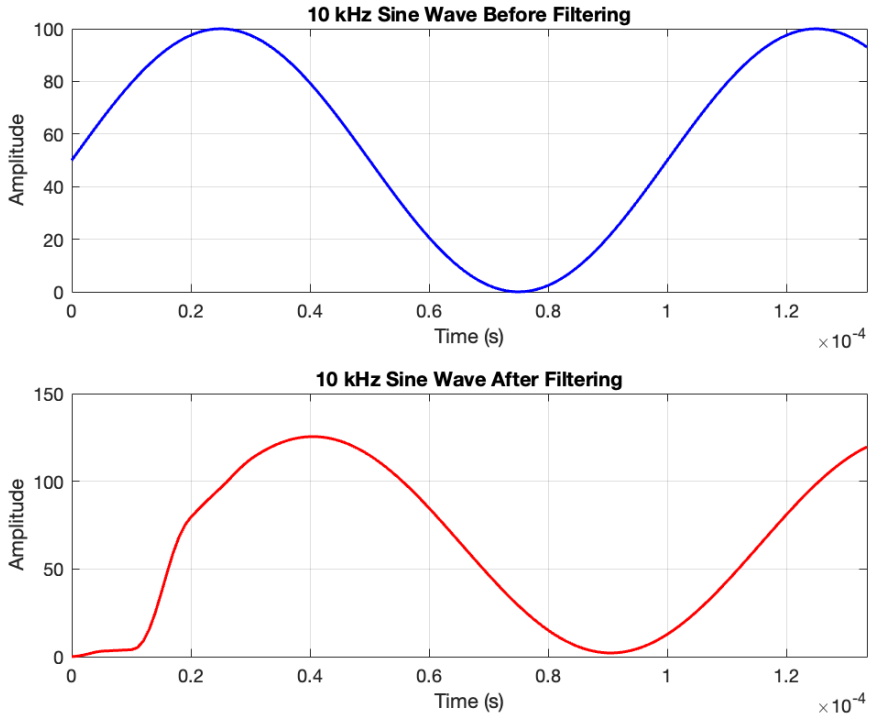

Figure 4: LPF Magnitude Response Without Negative coefficients

We can see that the magnitude response is considerably different than what we were getting originally. Which can affect our filtered output.

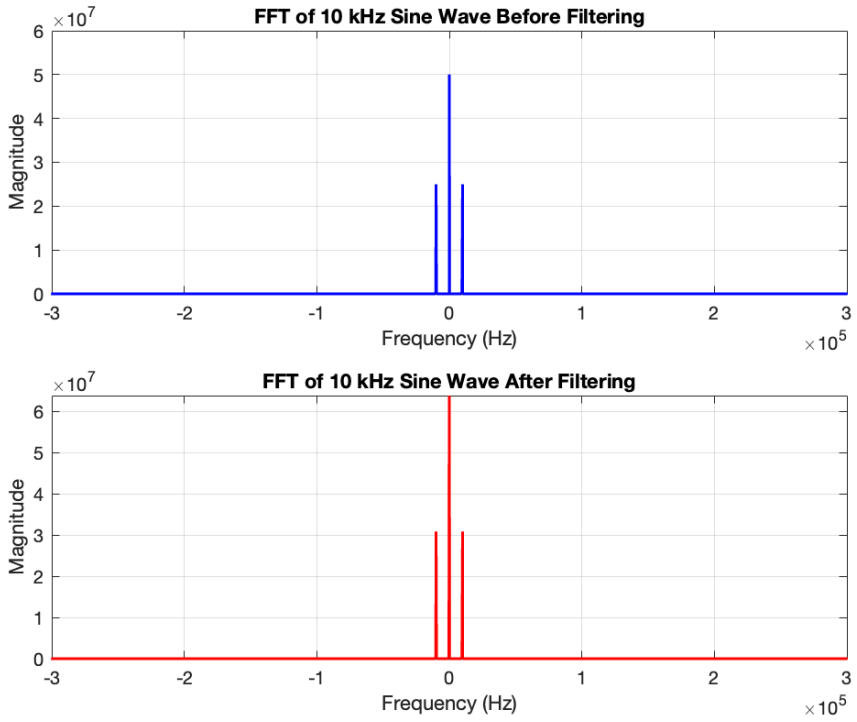
## 2.3 Testing the Filter on Matlab Generated Sine Waves

Will be testing for 10kHz, 50kHz, 200kHz and 250kHz frequencies.

### 2.3.1 10kHz Matlab Generated Sine Wave Results



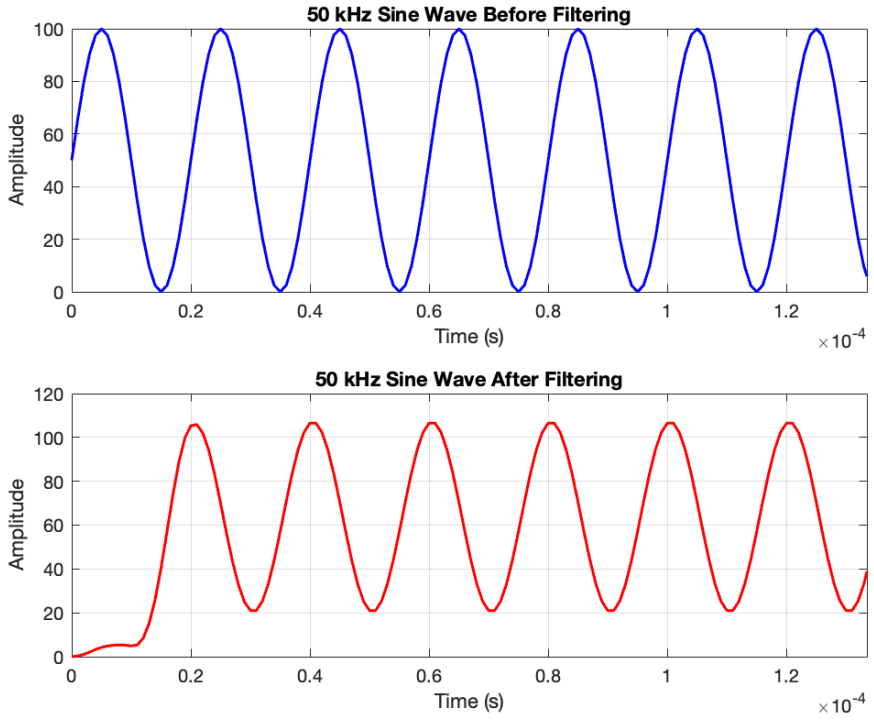

(a) 10kHz Sine Wave before and after being filtered



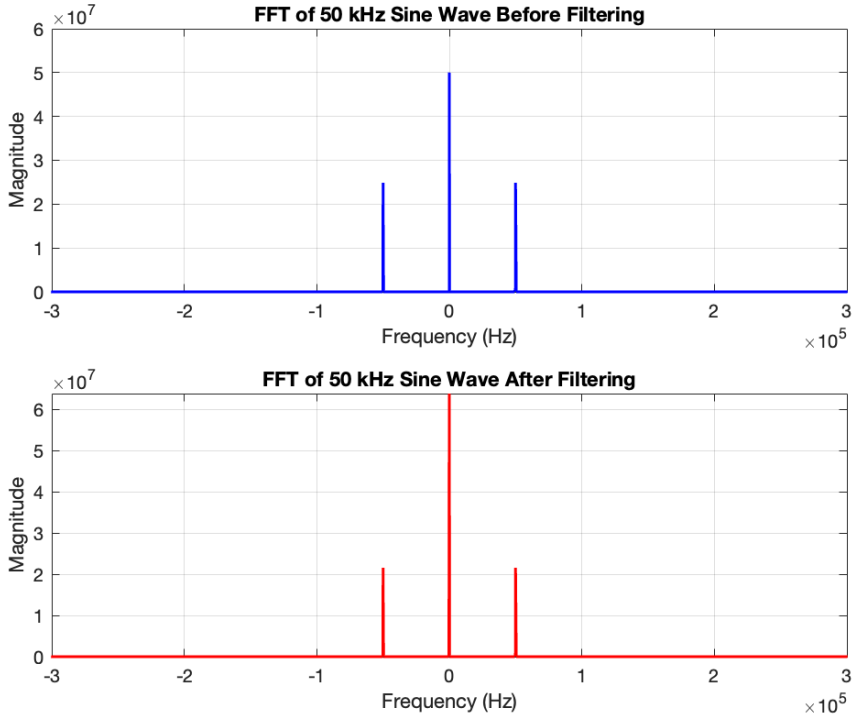

(b) 10kHz Sine Wave FFTs before and after being filtered

Figure 5: Filter Results on 10kHz Matlab Generated Sine Wave

### 2.3.2 50kHz Matlab Generated Sine Wave Results



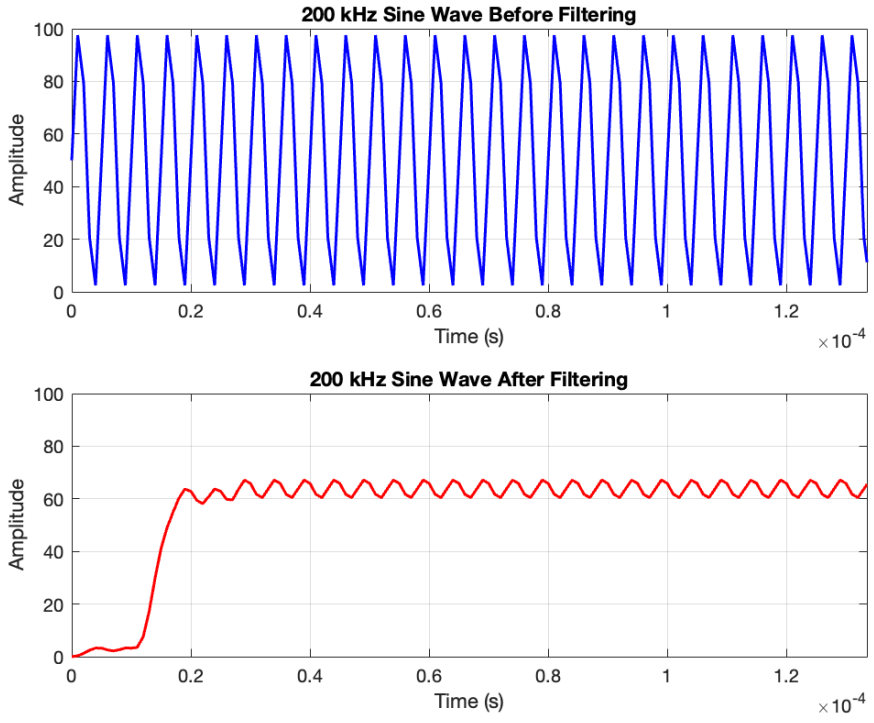

(a) 50kHz Sine Wave before and after being filtered



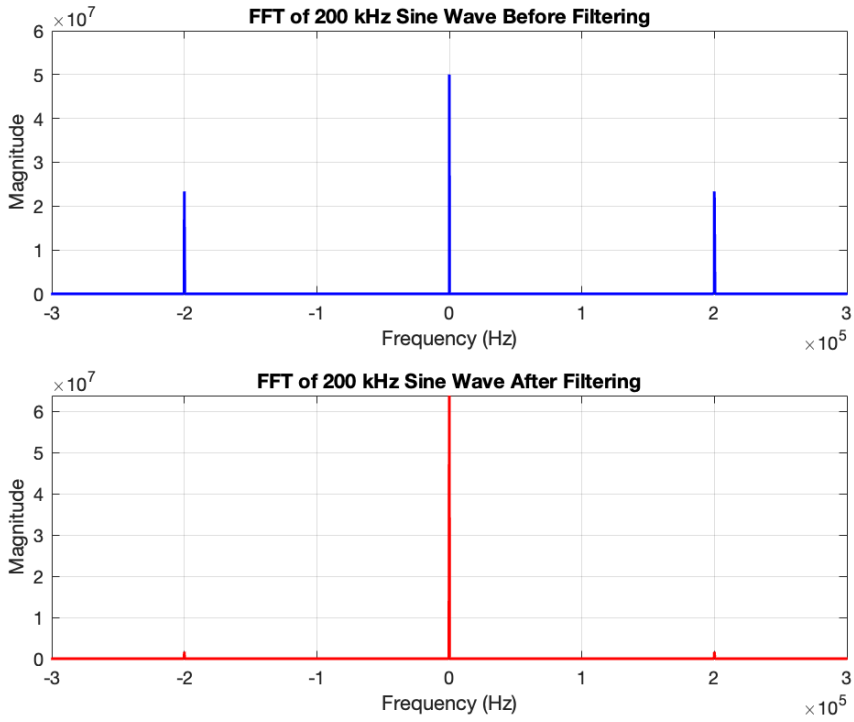

(b) 50kHz Sine Wave FFTs before and after being filtered

Figure 6: Filter Results on 50kHz Matlab Generated Sine Wave

### 2.3.3 200kHz Matlab Generated Sine Wave Results



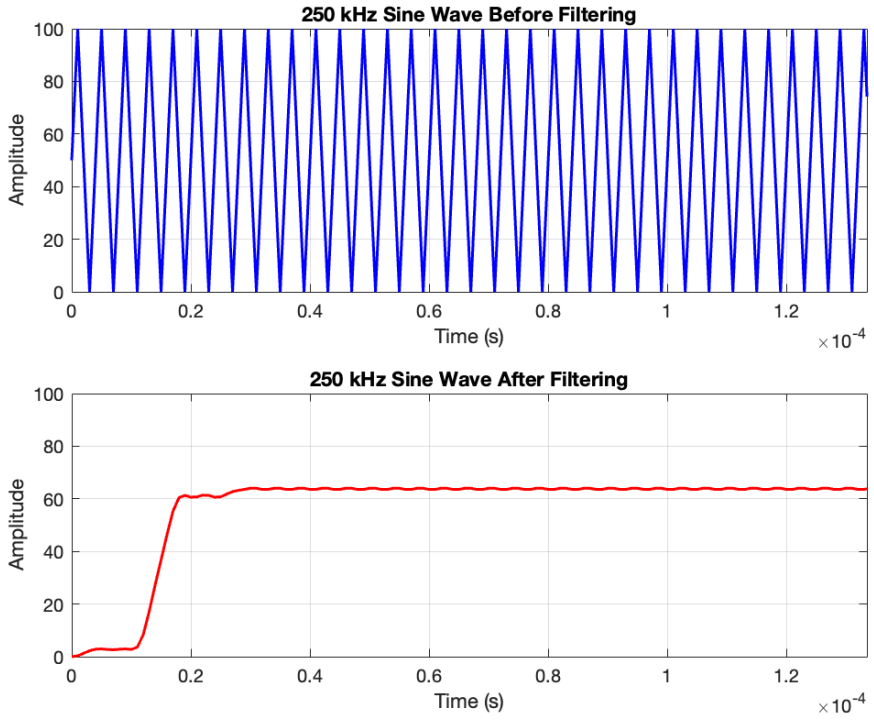

(a) 200kHz Sine Wave before and after being filtered



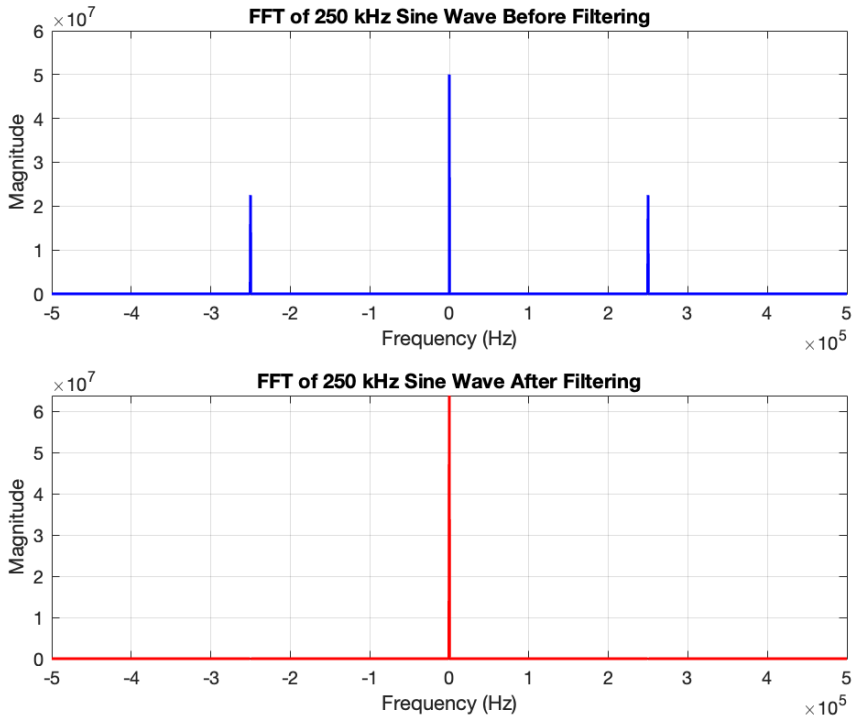

(b) 200kHz Sine Wave FFTs before and after being filtered

Figure 7: Filter Results on 200kHz Matlab Generated Sine Wave

### 2.3.4 250kHz Matlab Generated Sine Wave Results




(a) 250kHz Sine Wave before and after being filtered




(b) 250kHz Sine Wave FFTs before and after being filtered

Figure 8: Filter Results on 250kHz Matlab Generated Sine Wave

# 3 Converting Filter Coefficients to 8 Bit Binary Representation

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

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

0 2 3 4 4 2 0 0 0 0 0 5 18 32 44 50 50 44 32 18 5 0 0 0 0 0 2 4 4 3 2 0

# 4 Filter Implementation in Verilog

## 4.1 FIR Low Pass Filter:

### 4.1.1 Code:

```
`timescale 1ns/1ps

module fir_low_pass_filter(Data_out,
                           Data_in,
                           clk,
                           rst);

      parameter order = 32;
      parameter word_size_in = 8;
      parameter word_size_out = 2*word_size_in;

      parameter b0 = 8'd0;
      parameter b1 = 8'd2;
      parameter b2 = 8'd3;
```

```
15      parameter b3 = 8'd4;
16      parameter b4 = 8'd4;
17      parameter b5 = 8'd2;
18      parameter b6 = 8'd0;
19      parameter b7 = 8'd0;
20      parameter b8 = 8'd0;
21      parameter b9 = 8'd0;
22      parameter b10 = 8'd0;
23      parameter b11 = 8'd5;
24      parameter b12 = 8'd18;
25      parameter b13 = 8'd32;
26      parameter b14 = 8'd44;
27      parameter b15 = 8'd50;
28      parameter b16 = 8'd50;
29      parameter b17 = 8'd44;
30      parameter b18 = 8'd32;
31      parameter b19 = 8'd18;
32      parameter b20 = 8'd5;
33      parameter b21 = 8'd0;
34      parameter b22 = 8'd0;
35      parameter b23 = 8'd0;
36      parameter b24 = 8'd0;
37      parameter b25 = 8'd0;
38      parameter b26 = 8'd2;
39      parameter b27 = 8'd4;
40      parameter b28 = 8'd4;
41      parameter b29 = 8'd3;
42      parameter b30 = 8'd2;
43      parameter b31 = 8'd0;
44
45      output [word_size_out-1:0] Data_out;
46
47      input [word_size_in-1:0] Data_in;
48      input clk,rst;
49
50      reg [word_size_in-1:0] Samples [1:order];
51
52      integer k;
53
54      assign Data_out = b0 * Data_in +
55                        b1 * Samples[1] +
56                        b2 * Samples[2] +
57                        b3 * Samples[3] +
58                        b4 * Samples[4] +
59                        b5 * Samples[5] +
60                        b6 * Samples[6] +
61                        b7 * Samples[7] +
62                        b8 * Samples[8] +
63                        b9 * Samples[9] +
64                        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)
        if(rst == 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

endmodule
```

### 4.1.2 Schematic:

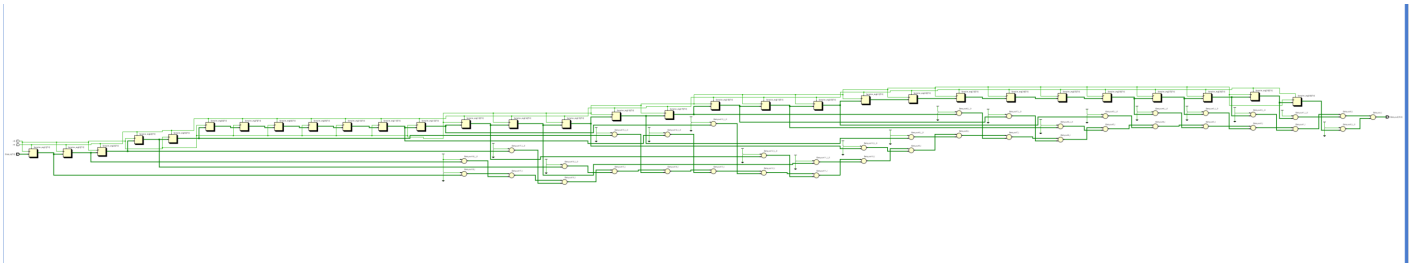


Figure 9: Low Pass Filter Schematic

## 4.2 Top Level Module:

This Combines the above filter with the two modules 'phase counter' and 'phase to amplitude' of the DDS

### 4.2.1 Code:

```
`timescale 1ns/1ps

module top_module (clk,
                   reset,
                   fsw,
                   fil_out);

    input clk, reset;
        input [9:0] fsw;

    output [15:0] fil_out;

    wire [9:0] dds_sin;
    wire [9:0] counter;

    // Instantiate phase counter and phase-to-amplitude converter
    phase_counter pc(.clk(clk),
                     .reset(reset),
                     .fsw(fsw),
                     .counter(counter));
    phase_to_amplitude pta(.counter(counter),
                           .reset(reset),
                           .dds_sin(dds_sin));

    // Instantiate low-pass FIR filter
    fir_low_pass_filter lpf(.Data_in(dds_sin),
                            .clk(clk),
                            .rst(reset),
                            .Data_out(fil_out));

endmodule
```

### 4.2.2 Schematic:

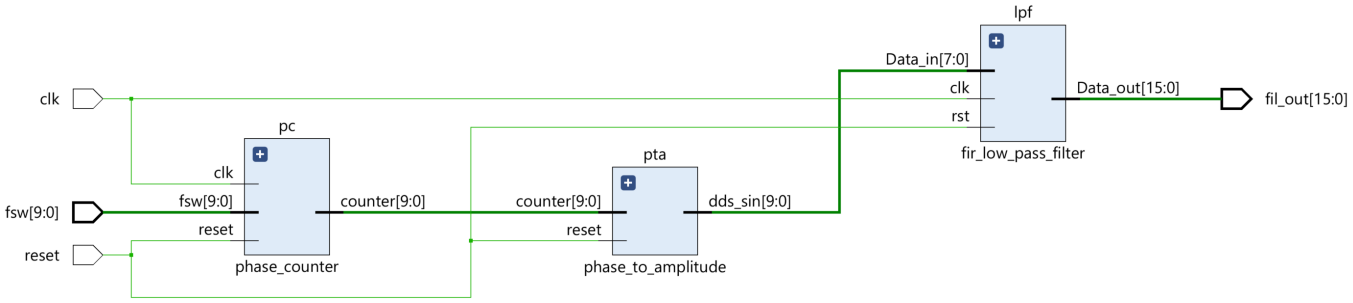



Figure 10: Top Level Schematic

## 4.3 Test-bench:

```
1  `timescale 1ns/1ps
2
3  `include "fir_lpf.v"
4  `include "phase_counter.v"
5  `include "phase_to_amplitude.v"
6  `include "top_module.v"
7
8  module tb_top_module();
9
10   reg clk;
11   reg reset;
12   reg [9:0] fsw;
13   wire [15:0] fil_out;
14
15   // Instantiate the top module
16   top_module uut(
17     .clk(clk),
18     .reset(reset),
19     .fsw(fsw),
20     .fil_out(fil_out)
21   );
22
23   // Clock generation (1 MHz)
24   initial begin
25     clk = 0;
26     forever #500 clk = ~clk; // 1 MHz clock period
27   end
28
29   // Reset generation
30   initial begin
```

```
31      $dumpfile("dump.vcd");
32      $dumpvars;
33      reset = 1;
34      #10 reset = 0; // Deassert reset after 10 time units
35    end
36
37    // Stimulus generation
38    initial begin
39      // Wait for a few clock cycles for initialization
40      #100;
41
42      // Apply frequency sweep of 10
43      fsw = 205;
44
45      // Simulate for a duration
46      #135500;
47
48      // Stop the simulation
49      $finish;
50    end
51
52    // Monitor for observing signals
53    always @(posedge clk) begin
54      $display("Time=%t,
55 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Counter=%d,
56 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣DDS_Sin=%d,
57 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Filtered_Output=%d",
58                 $time,
59                 uut.pc.counter,
60                 uut.pta.dds_sin,
61                 fil_out);
62    end
63
64  endmodule
```

# 5 Results

In verilog simulation I printed the time, counter, DDS sine wave amplitude values, and filtered output amplitude values in the logs. I then copied these output log values into a .txt file and used matlab to parse through it and generate one wave using dds output values and other using filtered output values, both against the time values. However for filtered output I also divided the values by 256 and then plotted them.

## 5.1 10kHz Sine Wave

I am adding the matlab script used for this, for just the 10kHz Sine Wave, as it is pretty much the same for other Sine Waves.

### 5.1.1 Simulation:

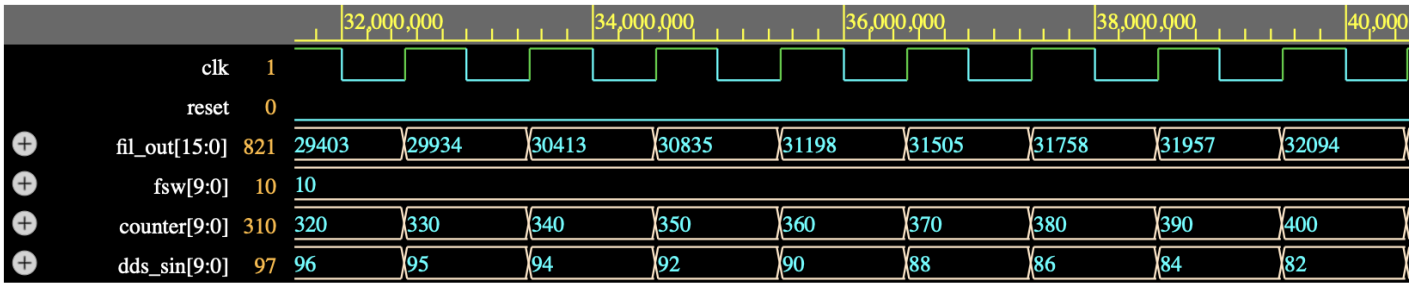



Figure 11: Simulation for 10kHz Sine Wave

### 5.1.2 Matlab Script Used For Plotting:

```
% Open the file for reading
fileID = fopen('10kHz_filtered.txt', 'r');
% Read the data using fscanf
data = fscanf(fileID, '# KERNEL: Time= %f, Counter= %d, DDS_Sin= %d,
   Filtered_Output=%d\n', [4, Inf]);
% Close the file
fclose(fileID);
% Extract counter, DDS_Sin, and time values
time = data(1, :);
counter = data(2, :);
dds_sin = data(3, :);
fil_out = data(4, :);
% Plot the graphs
subplot(2,1,1)
plot(time, dds_sin, "r", "LineWidth", 2);
hold on
% stairs(time, dds_sin, "b", "LineWidth", 2);
hold off
% Add labels and title
% legend('Analog', 'Digital');
xlabel('Time');
ylabel('Magnitude');
title('Original 10 kHZ Sine Wave');
% Show the grid
grid on;
subplot(2,1,2)
plot(time, (fil_out/256), "r", "LineWidth", 2);
hold on
% stairs(time, (fil_out/256), "b", "LineWidth", 2);
hold off
% Add labels and title
% legend('Analog', 'Digital');
xlabel('Time');
```

```
ylabel('Magnitude');
title('10 kHZ Sine Wave Filtered');
% Show the grid
grid on;
sgtitle("Comparison Between 10kHz Original & Filtered Output")
```

### 5.1.3 Plots:

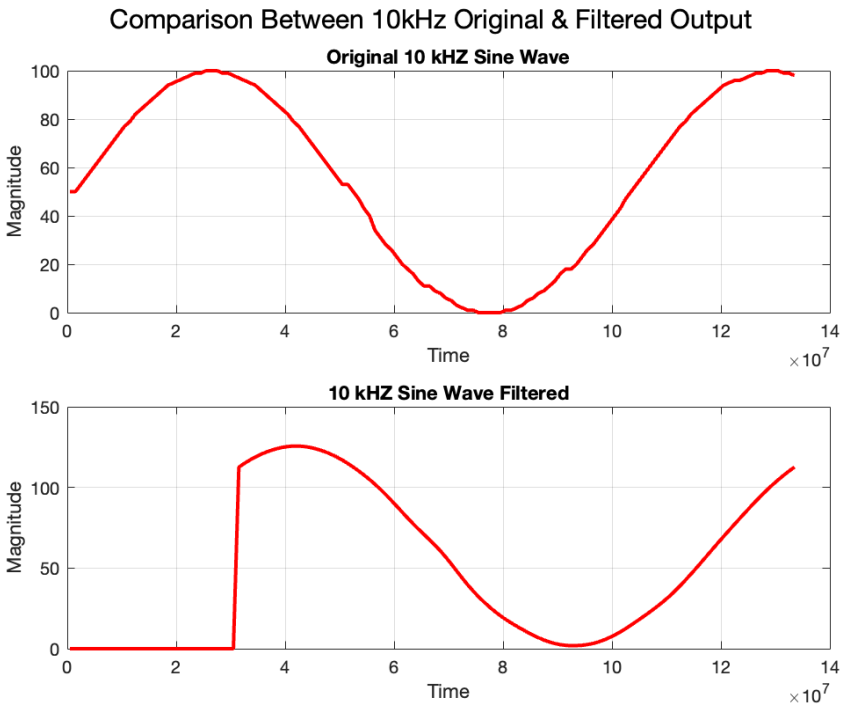



Figure 12: Filtered Results for 10kHz Sine Wave

## 5.2 50kHz Sine Wave

### 5.2.1 Simulation:

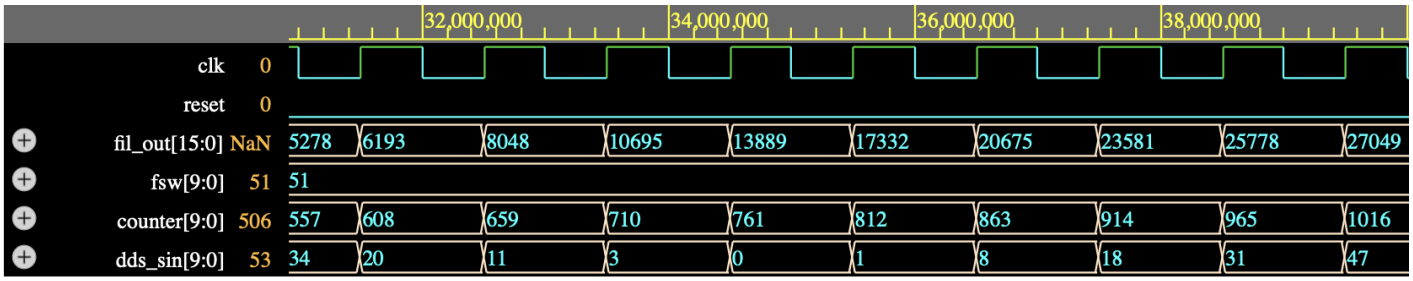



Figure 13: Simulation for 50kHz Sine Wave

### 5.2.2 Plots:

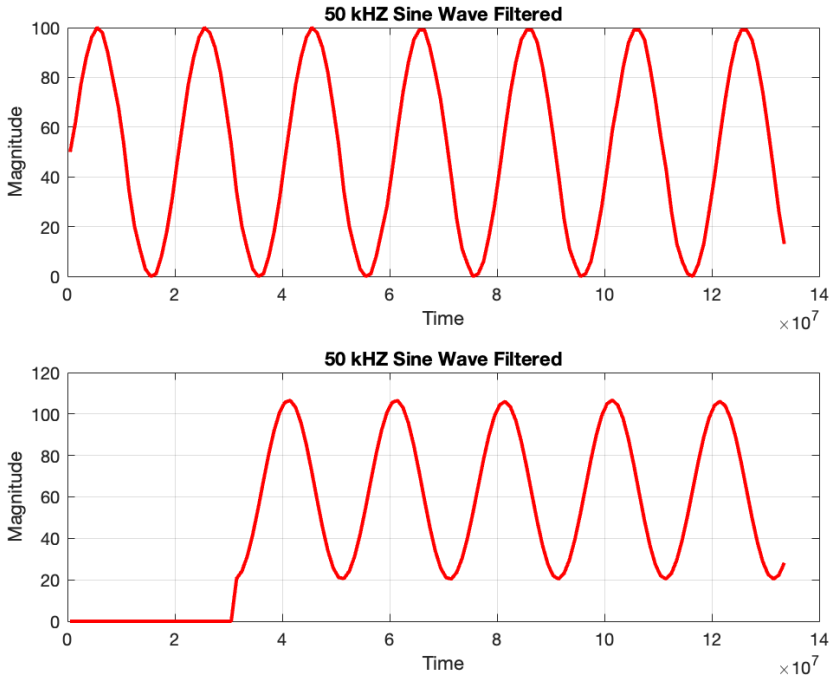



Figure 14: Filtered Results for 50kHz Sine Wave

## 5.3 200kHz Sine Wave

### 5.3.1 Simulation:

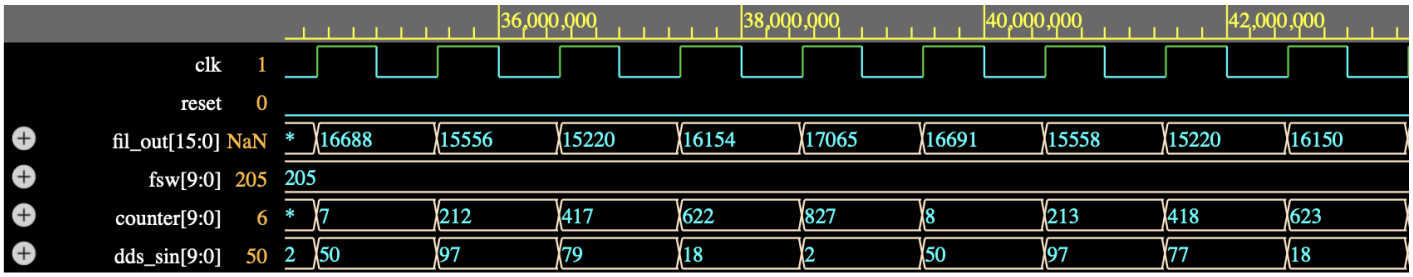



Figure 15: Simulation for 200kHz Sine Wave

### 5.3.2 Plots:

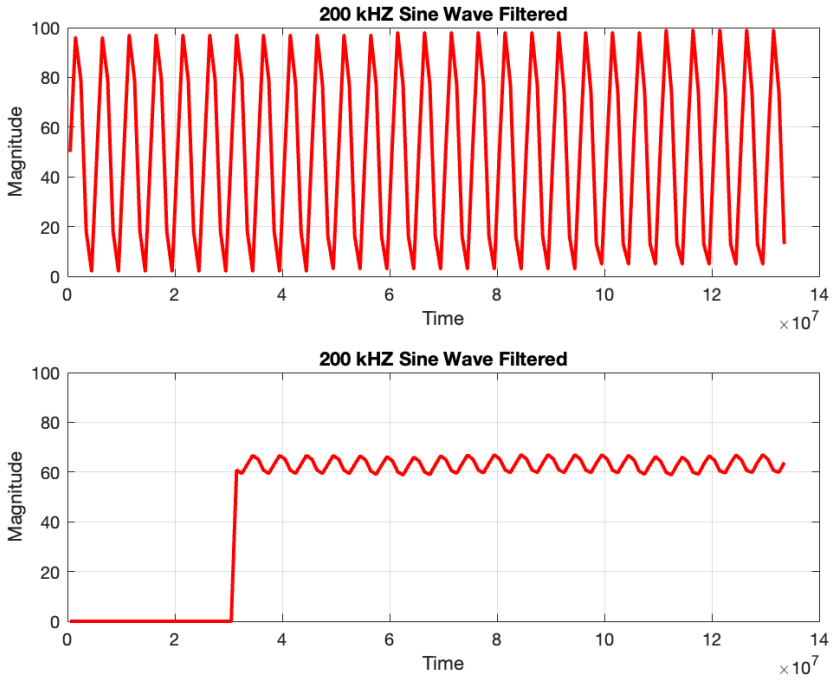



Figure 16: Filtered Results for 200kHz Sine Wave

## 5.4 250kHz Sine Wave

### 5.4.1 Simulation:

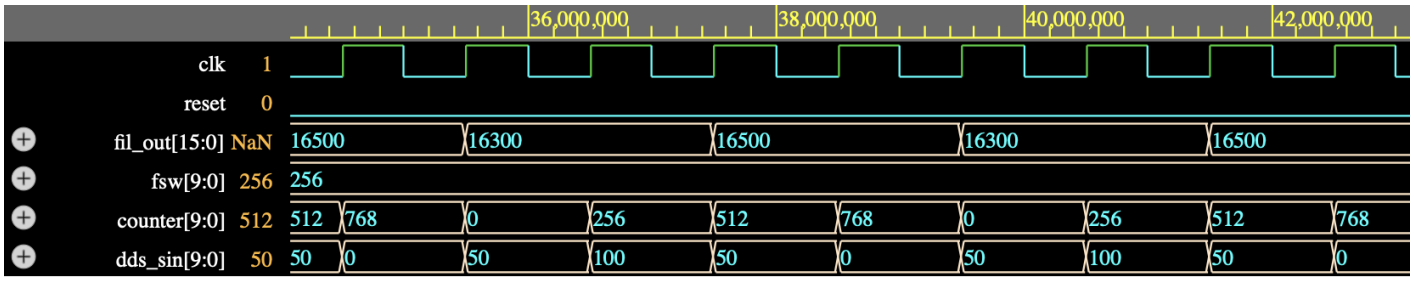



Figure 17: Simulation for 250kHz Sine Wave

### 5.4.2 Plots:

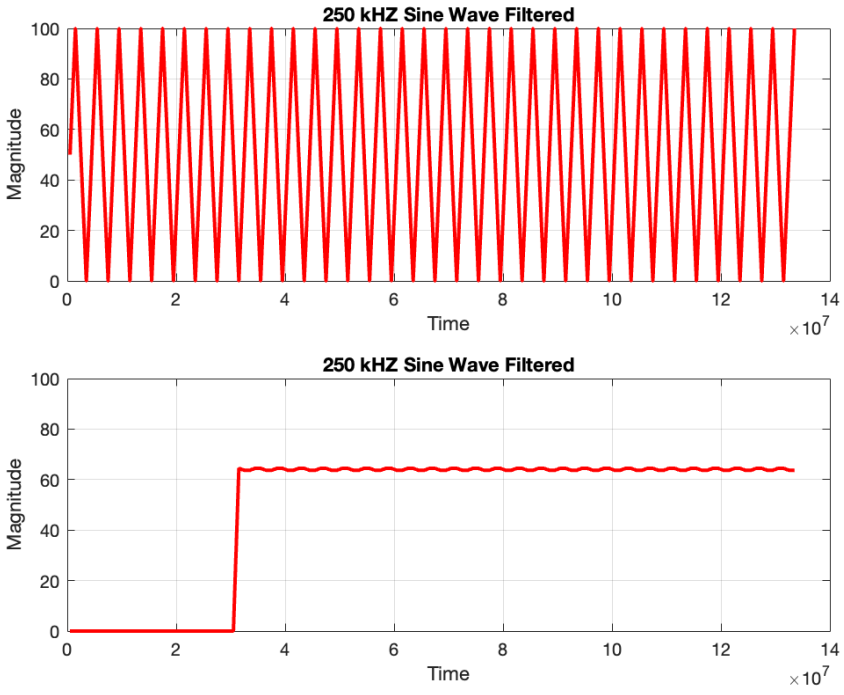



Figure 18: Filtered Results for 250kHz Sine Wave

In these Verilog Results you may notice a completely zero magnitude at the start. That is basically the period of don't care where the window is filling itself. For plotting purposes I have assumed don't care values to be zero. We can also notice a phase change in both Matlab and Verilog Results, whenever we have a filtered output. That is just a delay because of the characteristics of the filter we designed.