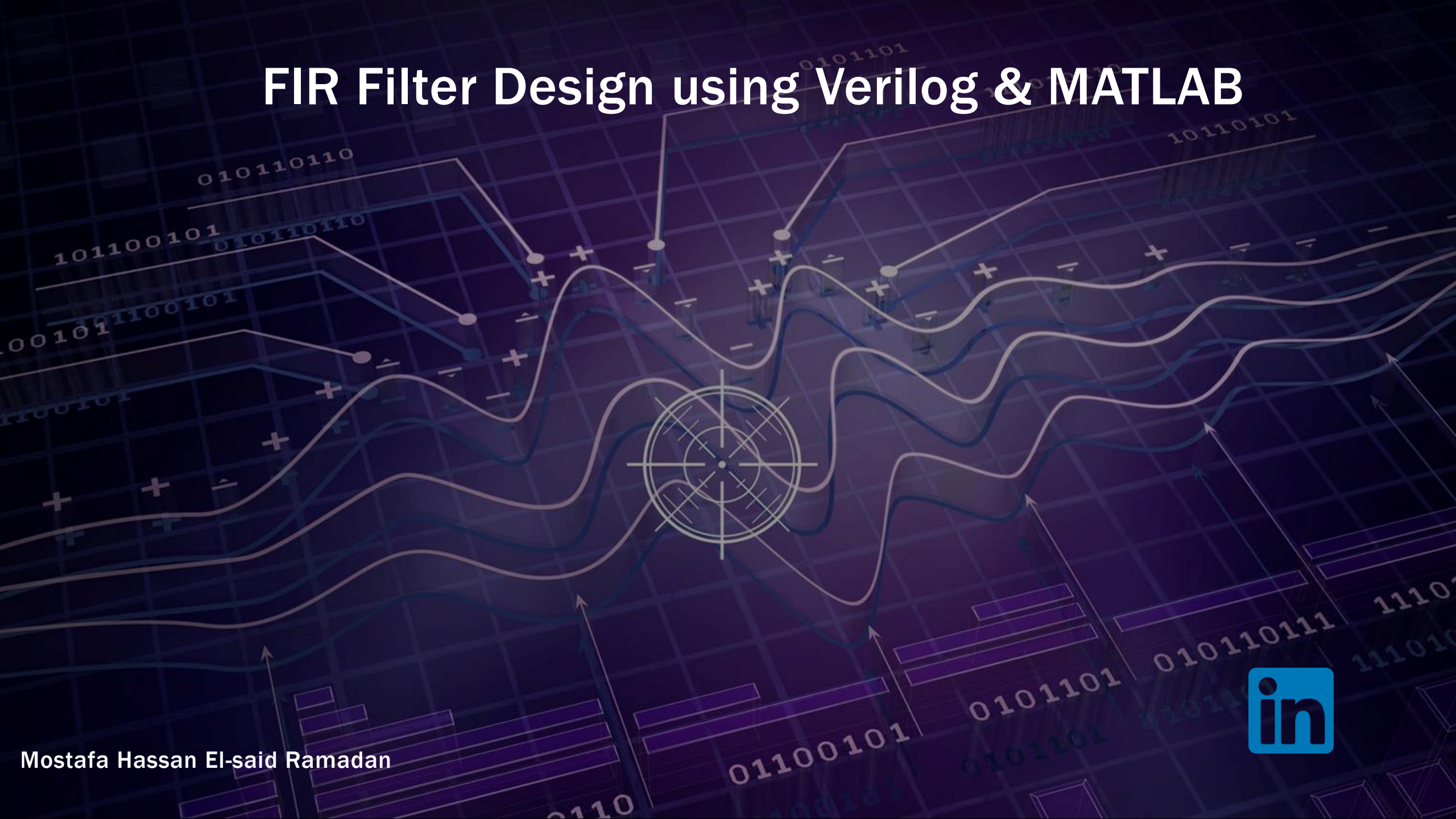
FIR Filter Design using Verilog & MATLAB



Mostafa Hassan El-said Ramadan

What is FIR filter ?

A FIR Filter is one of the most important building blocks in Digital signal processing. So it's important to learn how to design FIR filter and the theory of its operation.
A FIR filter is defined as a filter with an impulse response that settles to a zero over defined period of time (it has finite number of non-zero samples). The period of time for an FIR to settle is related to the order of the filter which is the order of the numerator of the transfer function of the FIR Filter(M).
One of the advantages of the FIR filter is that it has no feedback from the output, so the transfer function of the FIR Filter has no denominator (a_k=0) and it has the numerator coefficients as the impulse response($b_i = h[K]$).

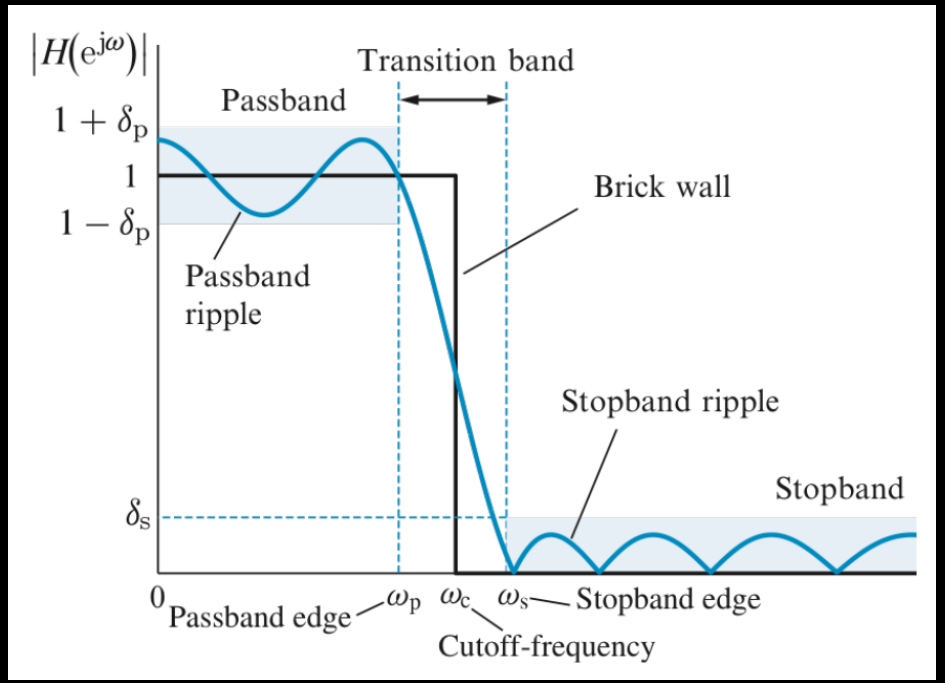
$$H(z) \equiv \frac{Y(z)}{X(z)} = \frac{\sum_{i=0}^{M} b_i z^{-i}}{1 + \sum_{k=1}^{N} a_k z^{-k}}$$

The algorithms used to design FIR filters are different from those used to design IIR filters.
The design of FIR filters is typically performed either in the discrete-time domain using windowing method or in the frequency domain using the frequency sampling method.

What is FIR filter ?

An ideal FIR Filter either pass or eliminate a region of the input spectrum and there's abrupt transition between pass band and stop band. But as we know ideal filters can't be implemented in real life, so we use an approximation to get the most practically realizable filter.

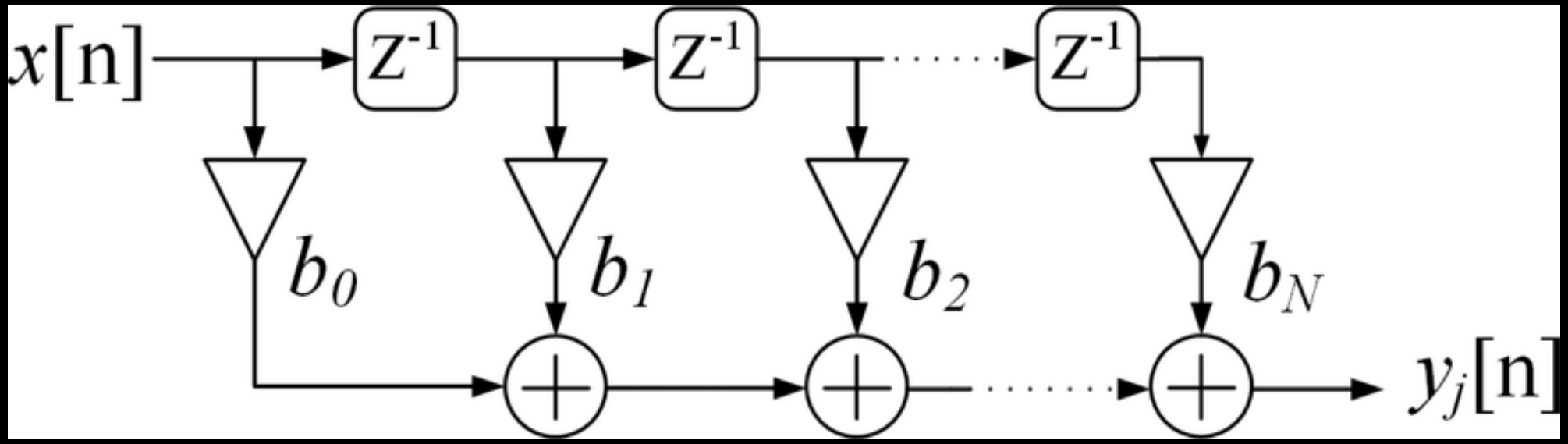
- For practical filter:
  1- the pass band response is not perfectly flat (it has passband ripples δ_p).
  2- the stop band can't completely reject the signal (it has stop band ripples δ_s). .
  3- there is a finite transition band between pass band and stop band.



FIR Filter structure

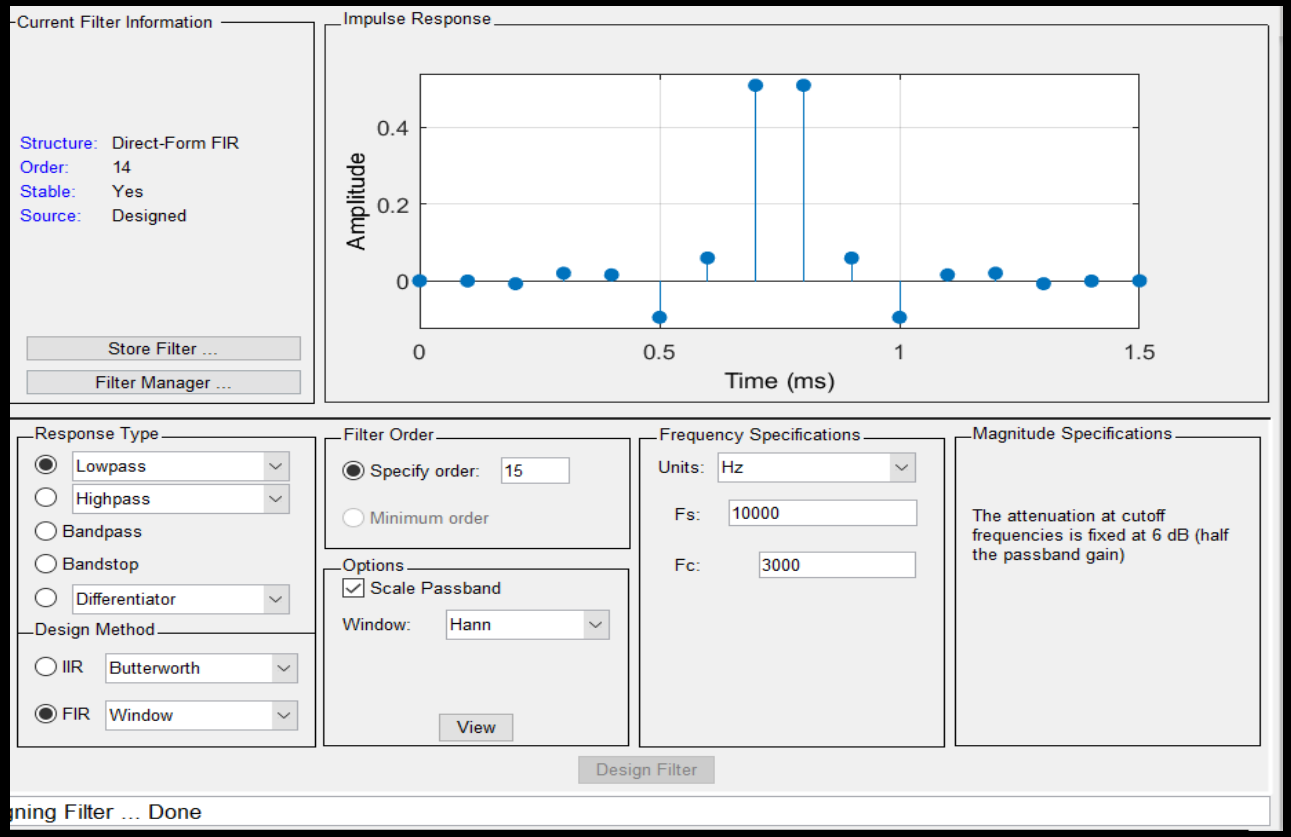
The main focus of this project is the implementation of a FIR in Verilog which can be break down into 4 main stages:

1- buffering stage to buffer the input data in a register.
2- shifting stage to shift the data in buffers.
2- multiplication stage to multiply the coefficients with the buffers.
4- accumulation stage to add the multiplied data.

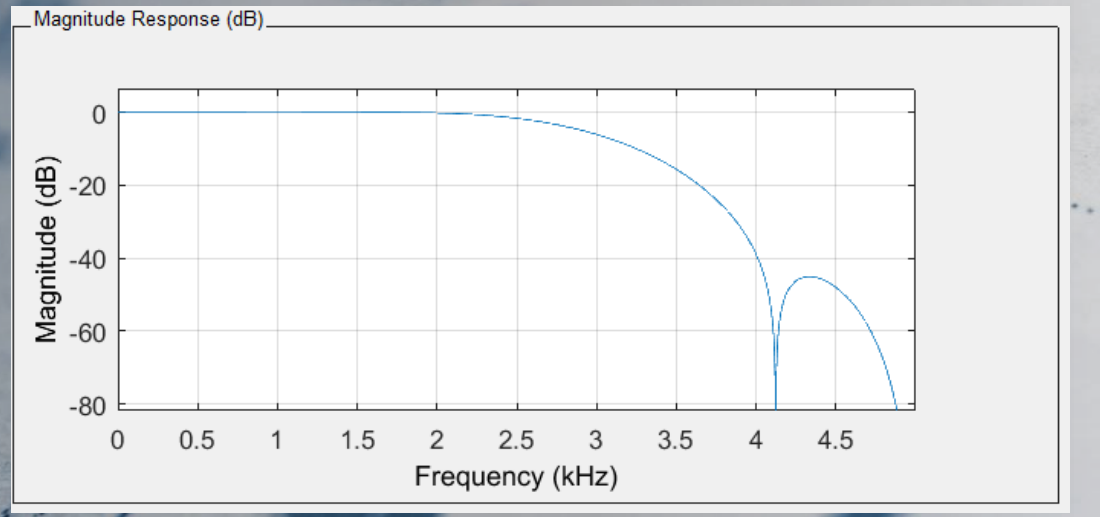


FIR Filter design using MATLAB

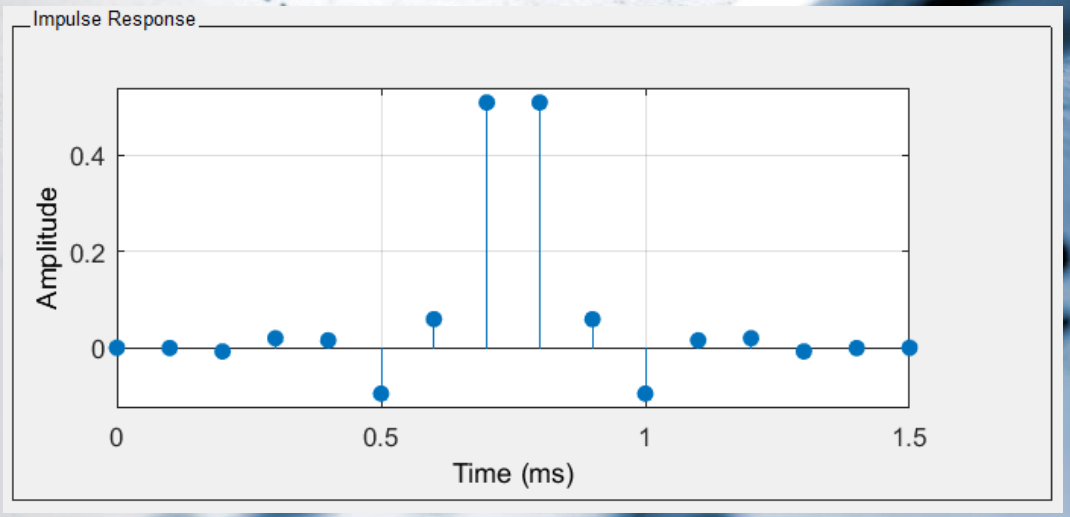
We will use FDA tool in MATLAB to estimate the filter coefficients by writing **fdatool** in the command window.
In this project we will design a 15th order Low pass FIR filter using Hann windowing method with cutoff frequency = 3 kHZ and sampling frequency = 10 kHZ.
We used Hann windowing method as it gives optimum response in the tradeoff between the smaller transition band ($\Delta\omega$) and the larger stop band attunation level (A_s).



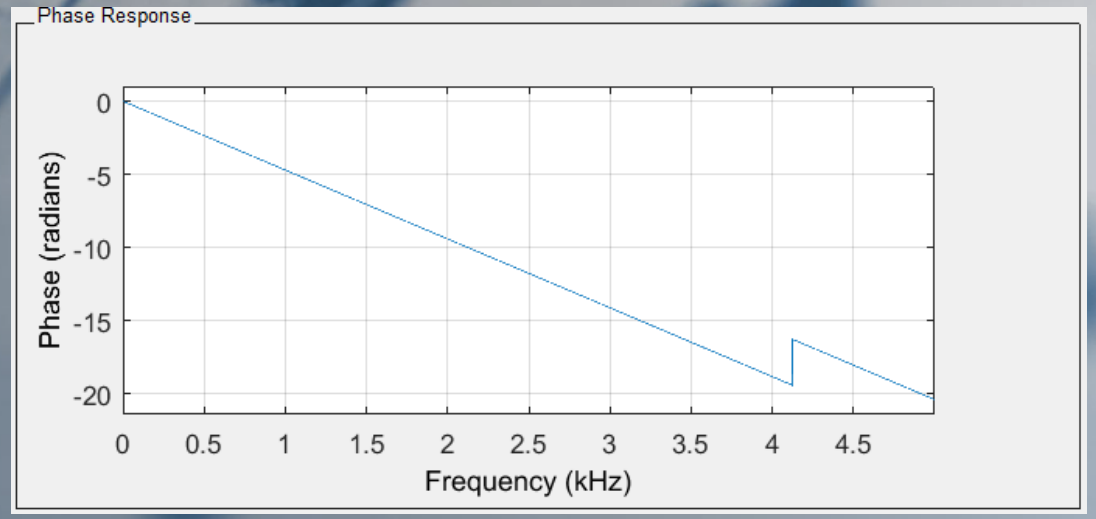
MATLAB plots



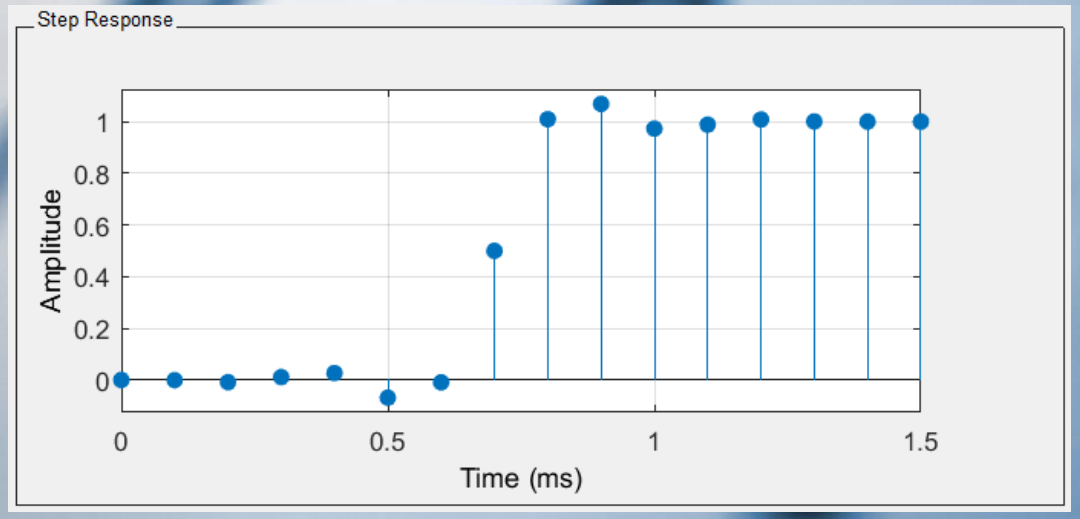
Magnitude response



Impulse response

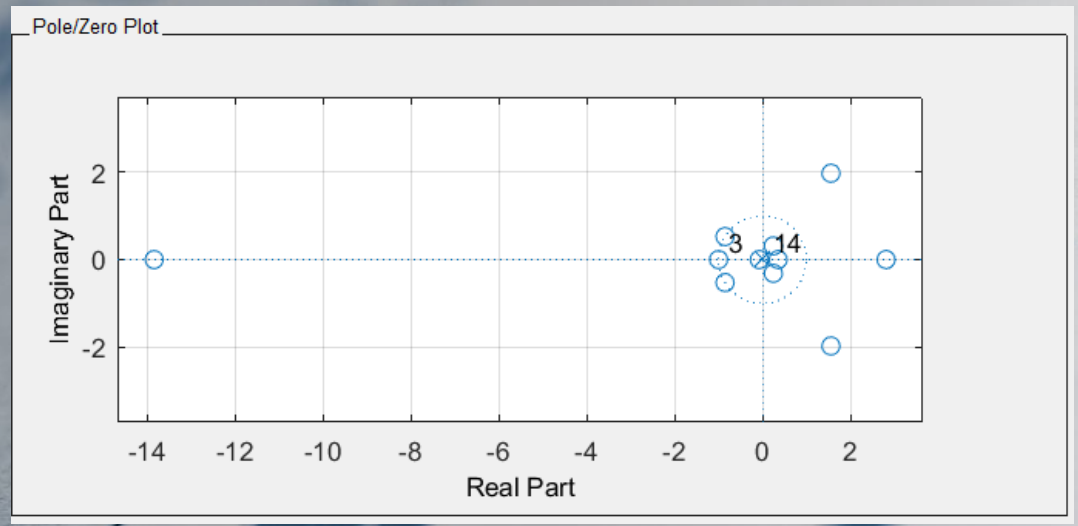


Phase response

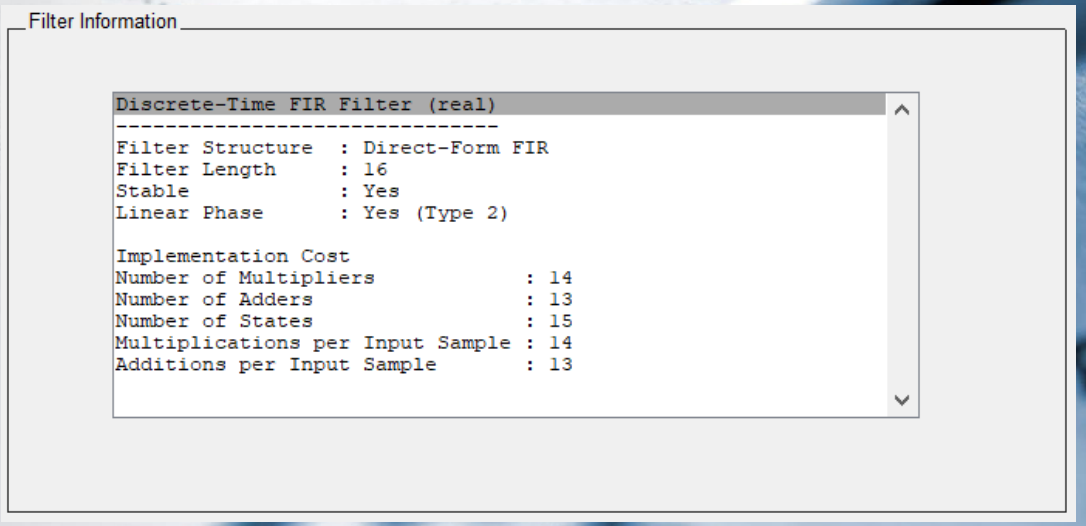


Step response

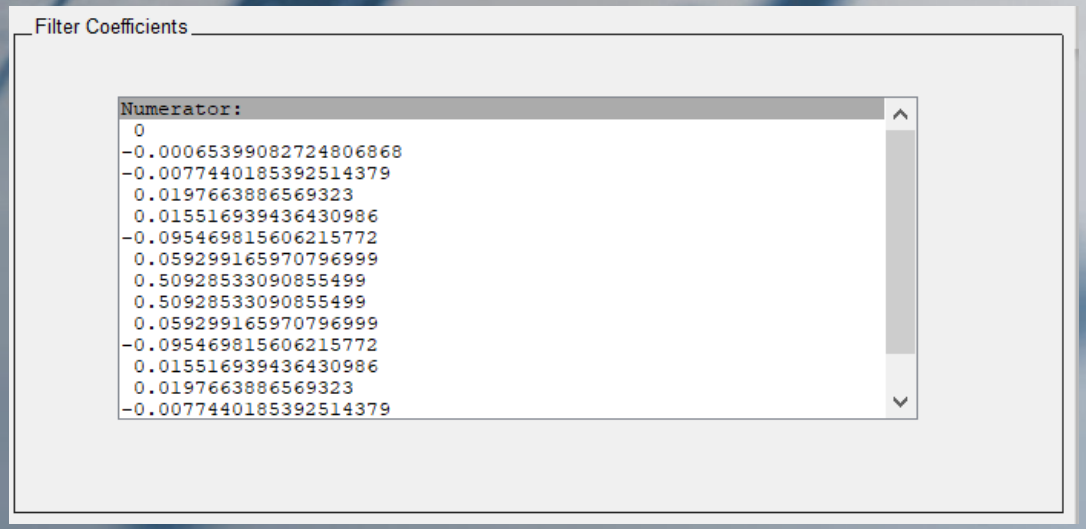
MATLAB plots



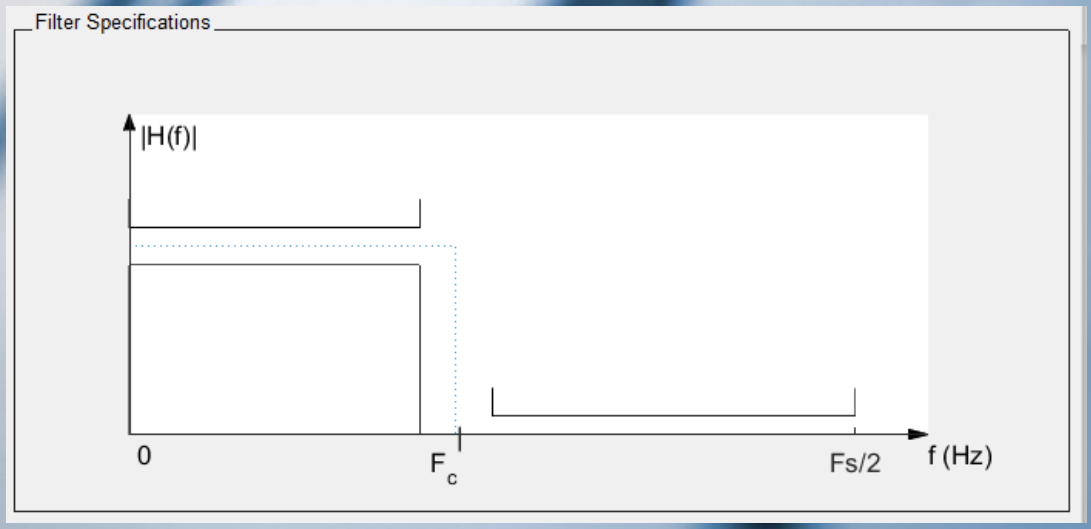
Poles & Zeros plot



Filter information



Filter coefficients



Filter Specifications

I/O signals documentation

Signal	Width	Type	Description
Clk	1	Input	System clock (100 kHZ)
reset	1	Input	Negative edge triggered asynchronous reset
Buff_en	1	Input	buffer enable
FIR_en	1	Input	Fir enable
FIR_data	16	Input	Unfiltered input data
FIR_data_filtered	32	Output	Filtered output data

Internal signals

```
//internal signals
wire signed [15:0] coeff0, coeff1,coeff2,coeff3,coeff4,coeff5,coeff6,coeff7,coeff8,coeff9,coeff10,coeff11,coeff12,coeff13,coeff14,coeff15 ;
reg signed [15:0] in_sample ;
reg signed [15:0] buff0, buff1,buff2,buff3,buff4,buff5,buff6,buff7,buff8,buff9,buff10,buff11,buff12,buff13,buff14,buff15 ;
reg signed [31:0] mul0, mul1,mul2,mul3,mul4,mul5,mul6,mul7,mul8,mul9,mul10,mul11,mul12,mul13,mul14,mul15 ;
```

FIR Filter design using Verilog

Now after we get the coefficients from MATLAB we will use it in our Verilog module by multiplying the coefficients By 2^{order} and covert the result to hexadecimal for good representation.
Since we used a 15th order FIR filter then:

$$coefficient = (fractional\ coefficient\ value) * 2^{15}$$

And then we round off the result (flooring) and convert it to hexadecimal.
For negative values we also round off the result and get the 2's compliment of the result and convert it to hexadecimal.

```
//FIR coefficient from matlab
assign coeff0 = 16'h0000; //0
assign coeff1 = 16'hFFEB; //-0.00065399082724806868
assign coeff2 = 16'hFF03; //-0.0077440185392514379
assign coeff3 = 16'h0287; //0.0197663886569323
assign coeff4 = 16'h01FC; //0.015516939436430986
assign coeff5 = 16'hF3C8; //-0.095469815606215772
assign coeff6 = 16'h0797; //0.059299165970796999
assign coeff7 = 16'h4130; //0.50928533090855499
assign coeff8 = 16'h4130; //0.50928533090855499
assign coeff9 = 16'h0797; //0.059299165970796999
assign coeff10 = 16'hF3C8; //-0.095469815606215772
assign coeff11 = 16'h01FC; //0.015516939436430986
assign coeff12 = 16'h0287; //0.0197663886569323
assign coeff13 = 16'hFF03; //-0.0077440185392514379
assign coeff14 = 16'hFFEB; //-0.00065399082724806868
assign coeff15 = 16'h0000; //0
```

FIR Filter design using Verilog

1- buffering stage

```
//buffering_stage
always@(posedge clk or negedge reset)
begin
  if(!reset)
    begin
      in_sample <= 16'h0 ;
    end
  else if(buff_en)
    begin
      in_sample <= fir_data ;
    end
  else
    begin
      in_sample <= in_sample ;
    end
end
```

FIR Filter design using Verilog

2- shifting stage

```
//shifting_stage
always@(posedge clk or negedge reset)
begin
  if(!reset)
    begin
      buff0 <= 16'b0 ;
      buff1 <= 16'b0 ;
      buff2 <= 16'b0 ;
      buff3 <= 16'b0 ;
      buff4 <= 16'b0 ;
      buff5 <= 16'b0 ;
      buff6 <= 16'b0 ;
      buff7 <= 16'b0 ;
      buff8 <= 16'b0 ;
      buff9 <= 16'b0 ;
      buff10 <= 16'b0 ;
      buff11 <= 16'b0 ;
      buff12 <= 16'b0 ;
      buff13 <= 16'b0 ;
      buff14 <= 16'b0 ;
      buff15 <= 16'b0 ;
    end
  else if(buff_en)
    begin
      buff0 <= in_sample ;
      buff1 <= buff0 ;
      buff2 <= buff1 ;
      buff3 <= buff2 ;
      buff4 <= buff3 ;
      buff5 <= buff4 ;
      buff6 <= buff5 ;
      buff7 <= buff6 ;
      buff8 <= buff7 ;
      buff9 <= buff8 ;
      buff10 <= buff9 ;
      buff11 <= buff10 ;
      buff12 <= buff11 ;
      buff13 <= buff12 ;
      buff14 <= buff13 ;
      buff15 <= buff14 ;
    end
```

FIR Filter design using Verilog

3- multiplication stage

```
//multiplication_stage
always@(posedge clk or negedge reset)
begin
  if(!reset)
    begin
      mul0 <= 32'b0 ;
      mul1 <= 32'b0 ;
      mul2 <= 32'b0 ;
      mul3 <= 32'b0 ;
      mul4 <= 32'b0 ;
      mul5 <= 32'b0 ;
      mul6 <= 32'b0 ;
      mul7 <= 32'b0 ;
      mul8 <= 32'b0 ;
      mul9 <= 32'b0 ;
      mul10 <= 32'b0 ;
      mul11 <= 32'b0 ;
      mul12 <= 32'b0 ;
      mul13 <= 32'b0 ;
      mul14 <= 32'b0 ;
      mul15 <= 32'b0 ;
    end
  else if(fir_en)
    begin
      mul0 <= coeff0 * buff0 ;
      mul1 <= coeff1 * buff1  ;
      mul2 <= coeff2 * buff2  ;
      mul3 <= coeff3 * buff3 ;
      mul4 <= coeff4 * buff4  ;
      mul5 <= coeff5 * buff5 ;
      mul6 <= coeff6 * buff6  ;
      mul7 <= coeff7 * buff7  ;
      mul8 <= coeff8 * buff8  ;
      mul9 <= coeff9 * buff9  ;
      mul10 <= coeff10 * buff10  ;
      mul11 <= coeff11 * buff11 ;
      mul12 <= coeff12 * buff12  ;
      mul13 <= coeff13 * buff13  ;
      mul14 <= coeff14 * buff14 ;
      mul15 <= coeff15 * buff15 ;
    end
  end
```

FIR Filter design using Verilog

4- accumulation stage

```
//adding_stage
always@(posedge clk or negedge reset)
begin
  if(!reset)
    begin
      fir_filtered_data <= 32'b0;
    end
  else if(fir_en)
    begin
      fir_filtered_data <= mul0 + mul1 + mul2 + mul3 + mul4 + mul5 + mul6 + mul7 + mul8 + mul9 + mul10 + mul11 + mul12 + mul13 + mul14 + mul15 ;
    end
end
```

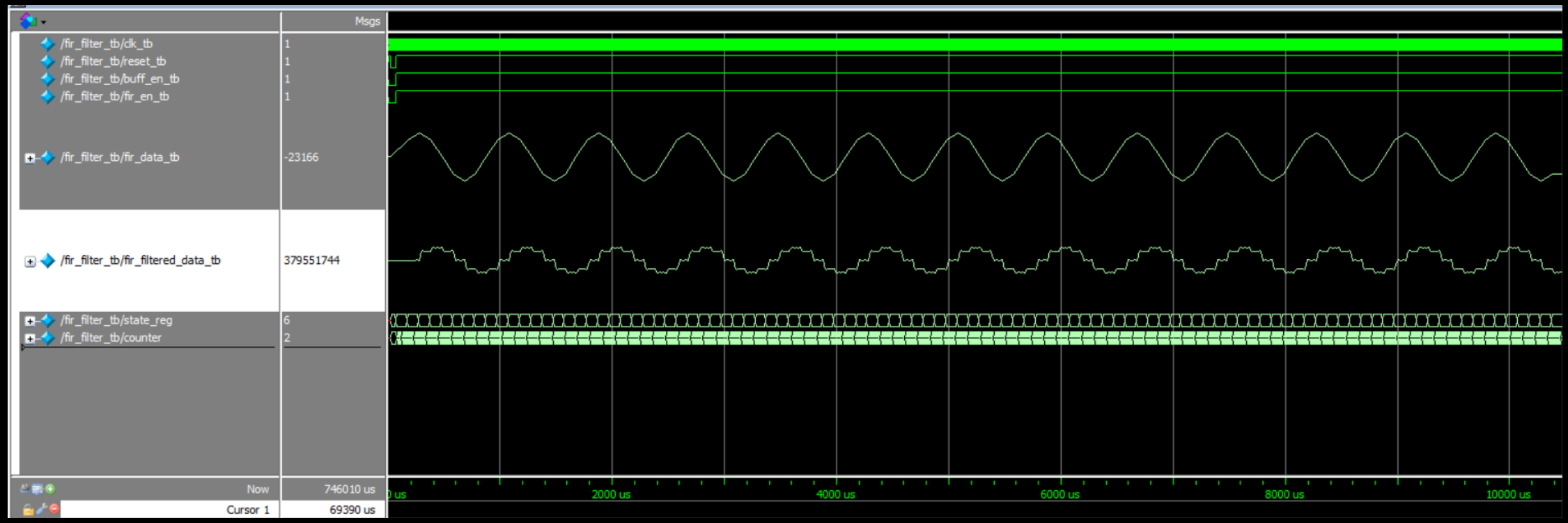
FIR Filter testbench

We will use a sine wave as an input signal to the filter and we will change the frequency of the sine wave to observe the output signal.
First we reset the Filter and then start to apply the sine signal.

```
initial
 begin
        reset_tb = 1;
        buff_en_tb = 0 ;
        fir_en_tb = 0 ;
        #20;
        reset_tb = 0;
        buff_en_tb = 0 ;
        fir_en_tb = 0 ;
        #50;
        reset_tb = 1;
        buff_en_tb = 1;
        fir_en_tb = 1;
        #1000000;
        $stop ;
 end
```

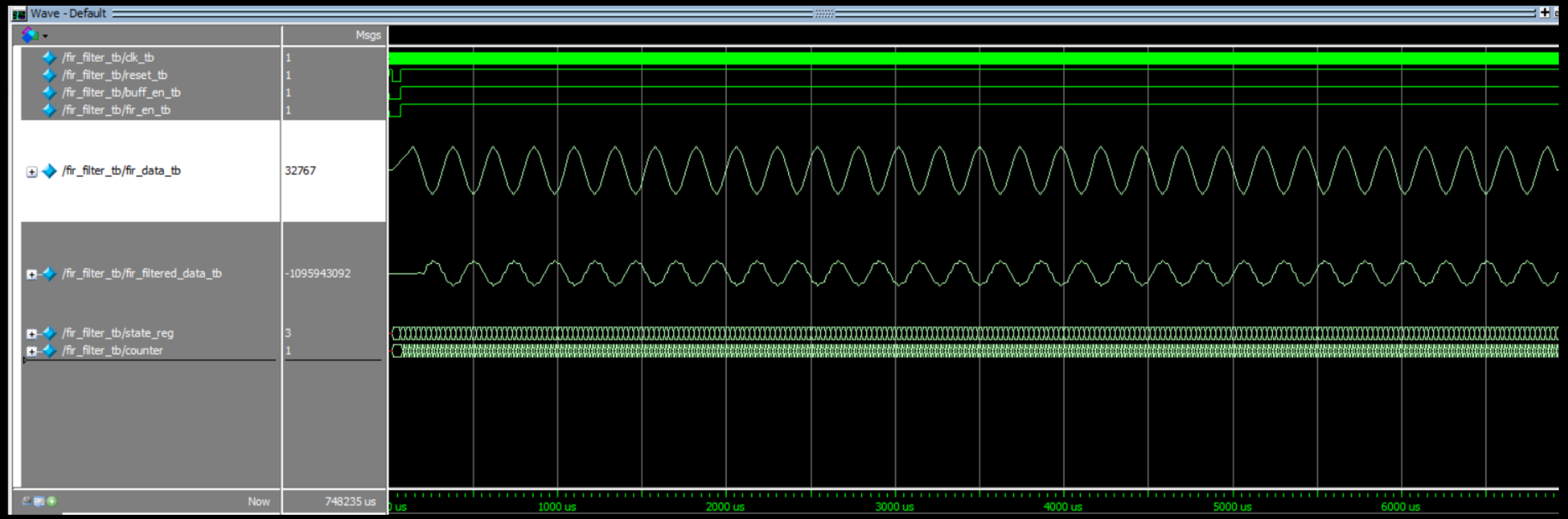
FIR Filter waveforms

1.25 kHZ sine wave input (below the cutoff frequency)



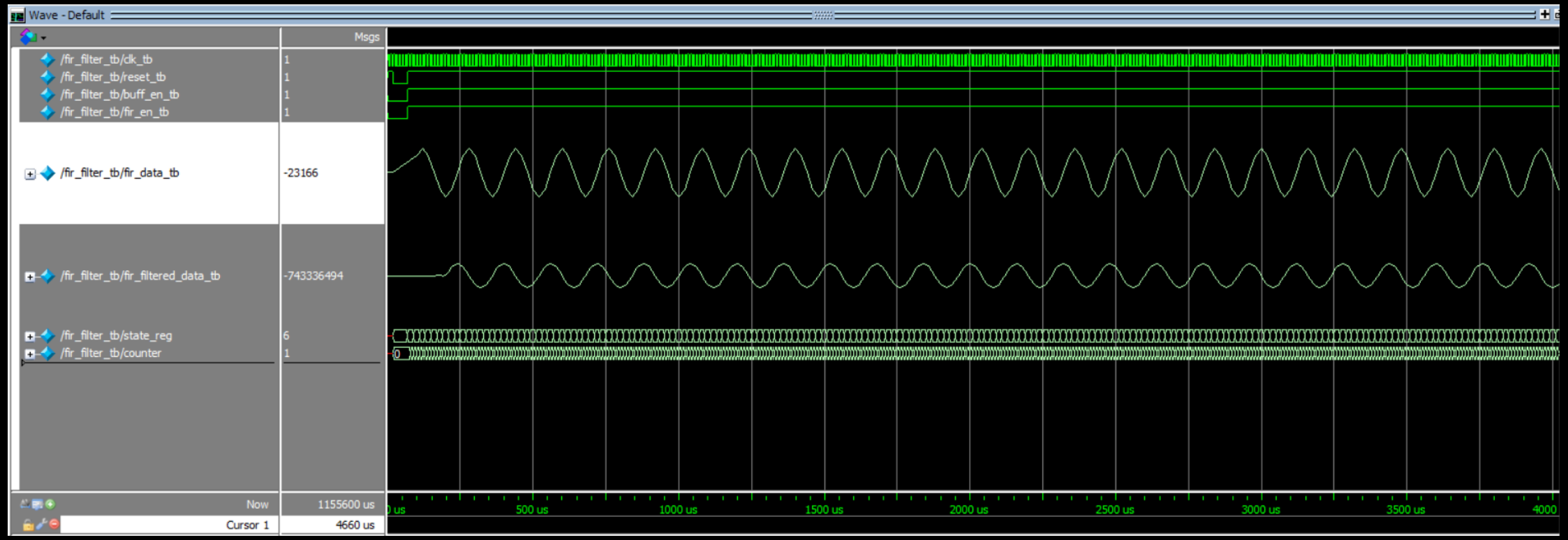
FIR Filter waveforms

4.167 kHZ sine wave input (above the cutoff frequency)



FIR Filter waveforms

6.25 kHZ sine wave input (above the cutoff frequency)



For the design code and the testbench code : https://github.com/MostafaHassan0053/FIR_Filter

Thank you!