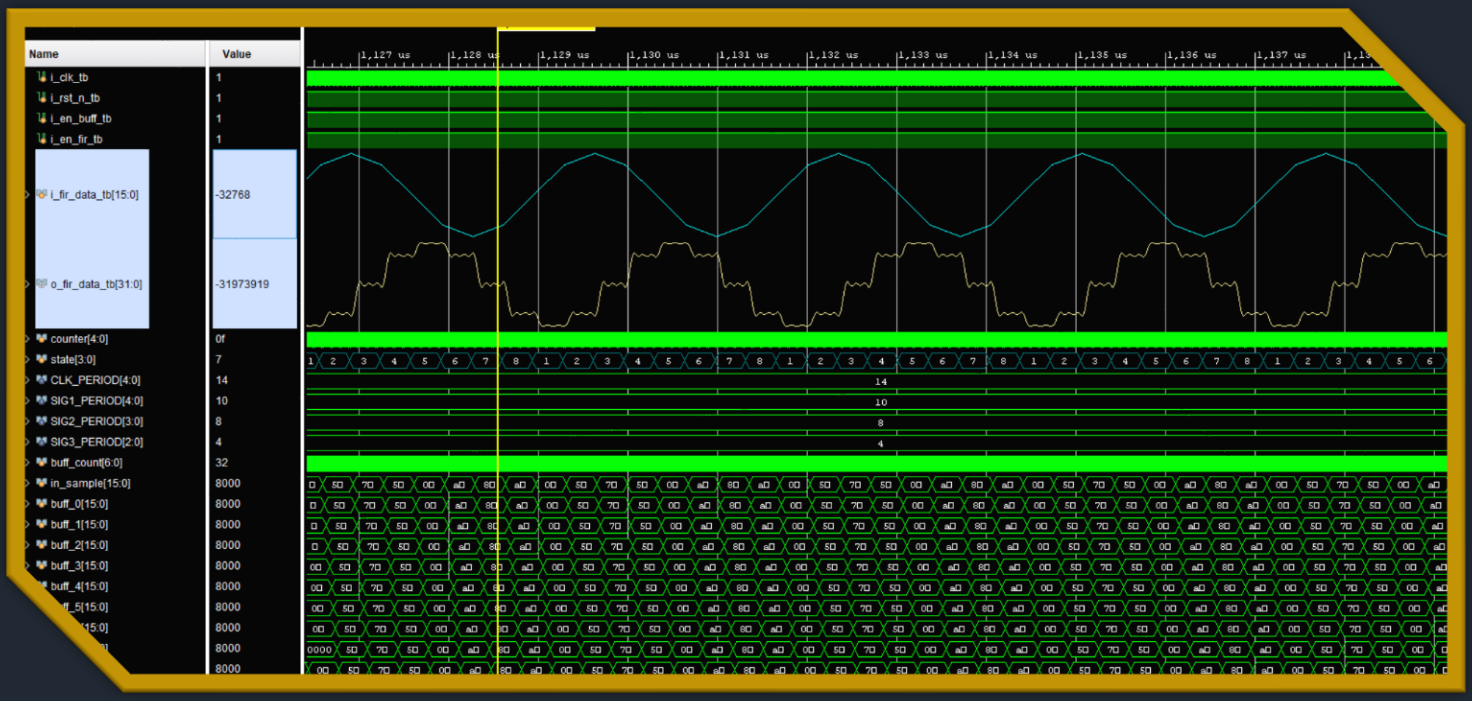


FIR Filter

Digital Design Using Verilog HDL

Contents:

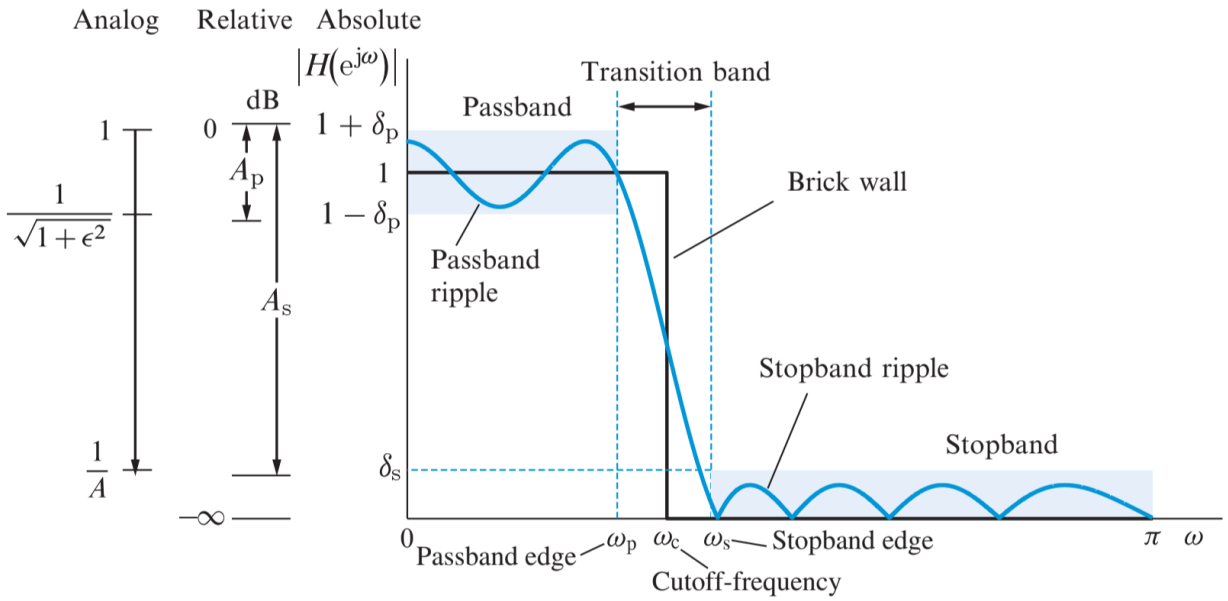
1. Theoretical Background

Filters are one of the most basic Building Blocks in DSP (whether it's for an ASIC or to be used on FPGA), They are vital blocks in any system.

Mainly, A Filter is a **LTI system** that is used as a **Signal Conditioning** block which main role is to select the range of frequencies that will be either filtered out or allowed to pass through and it has 4 main types:

a) **Low Pass Filter (LPF):** allows frequencies that are below a certain threshold (Fc) to pass while attenuating above frequencies.
b) **High Pass Filter (HPF):** allows frequencies that are above a certain threshold (Fc) to pass while attenuating below frequencies.
c) **Band Pass Filter (BPF):** allows frequencies that are between two limits (Fc1 & Fc2) to pass while attenuating other frequencies.
d) **Band Reject (Stop) Filter:** attenuates frequencies that are between two limits (Fc1 & Fc2) while allowing other frequencies.

For our Design, we will study **FIR Low Pass Filters:**



Example of tolerance diagram for a lowpass filter, Applied Digital Signal Processing reference.

- **FIR Filters**

FIR filter is an Impulse response that settles at zero after a finite period of time, hence called a Finite Impulse Response

$$y[n] = \sum_{k=0}^{M} b_k \, x[n-k]$$

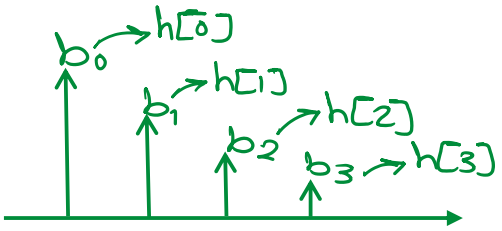
- $x[n-k]$: input sampled signal (discrete)
- $y[n]$: The Filtered signal (FIR output)
- b_k : Filter Coefficients, which equals the impulse response.

Let input signal be impulse to get the impulse response: ***Proof***

$$x[n-k] = \delta[n-k] \rightarrow y[n] = h[n]$$

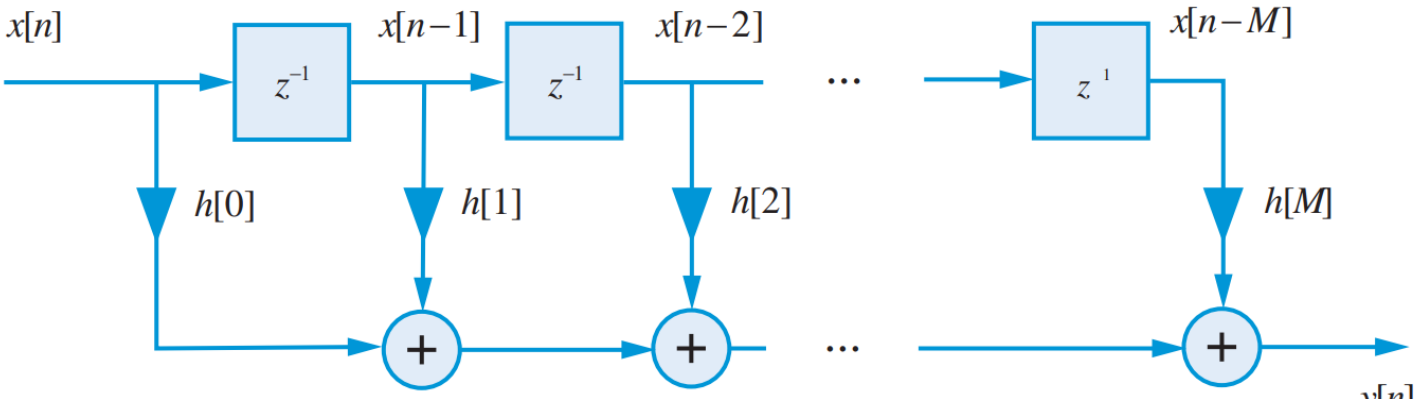
$$\therefore h[n] = \sum_{k=0}^{M} b_k \, \delta[n-k] \rightarrow$$

$$\therefore y[n] = \sum_{k=0}^{M} b_k \, x[n-k] = \sum_{k=0}^{M} h[n] \, x[n-k]$$



Here, filter coefficients b_k are **feed-forward** coefficients as there is no feed-back in FIR filters, unlike the IIR Filter which has feed-back, hence, knowing all the above, if the input is a single impulse, output will be the filter coefficient. $y[n_o] = b_k$

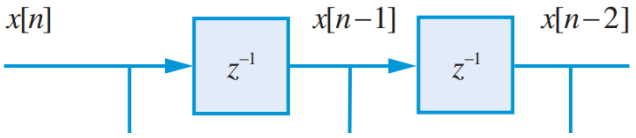
So, this mentioned equation translates to the following Block:



Block diagram representation of an FIR system, Applied Digital Signal Processing reference.

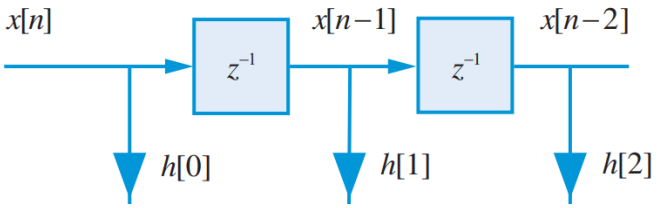
Understanding FIR Filter Block Diagram:

a) **Z-Transform (Delay Blocks)**



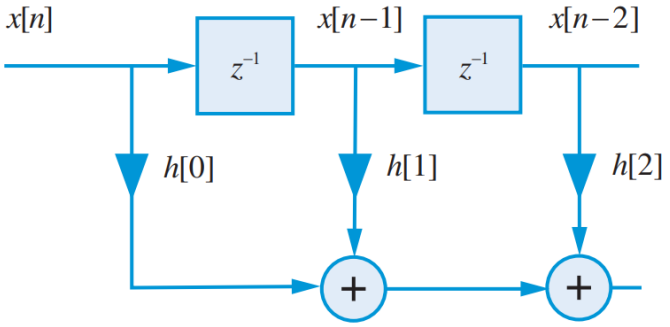
Here, signal $x[n]$ is delayed by a single time step after the first delay block and then delayed again and again and so on, this represents the term $x[n-k]$ inside the summation.

b) **Impulse Response (Gain)**



Here, each delayed signal is multiplied by the impulse response, this represents the term $h[n]$ inside the summation that $x[n-k]$ is being multiplied by.

c) **Accumulation**



Here, the results of the multiplication of each stage is summed and accumulated to the filter output signal $y[n]$

Before we start designing our filter, we need to understand Filter Taps

Taps: The impulse response of the filter is called a filter tap, which represents the filter coefficients, when saying **N-Tap filter**, we mean the number of memory needed to implement the filter, of course we don't mean a memory that captures the output and feed it top the input as FIRs are feed-forward only, but we mean it as a register file that saves the filter coefficient values, as if we are designing a 50-Tap Filter we will need 50 place in a Memory to save the coefficients values that the input signal is being multiplied by.

Higher taps mean higher frequency resolution, hence narrower filter and steeper roll off which in turns allow for a sharper filter response, of course all of that comes with a cost, and the cost here is the delay and complexity, as more taps means higher order which will add more delay.

2. Filter Design

FIR Design will go by three main blocks:

1) Circular buffers: represents the Z-transform block which will clock each samples delay properly.
2) Multipliers: for each of the taps coefficient values (impulse response) to be multiplied by the input samples.
3) Accumulator register: for summing the results of the multiplication processes.

Choosing the Filter to be Designed:

We will follow **Widowing method** in designing our FIR Filter, and our chosen window will be **Hamming Window**, so our design parameters will go as follows:

> **A 66-order (67 taps) hamming FIR LPF Filter is to be designed with a cutoff frequency of 200 KHZ for a sampling frequency of 1 MHZ**

So, we will start our design by the MATLAB stage, to get our chosen filter coefficients (impulse response)

a) Generating Filter Coefficients using MATLAB

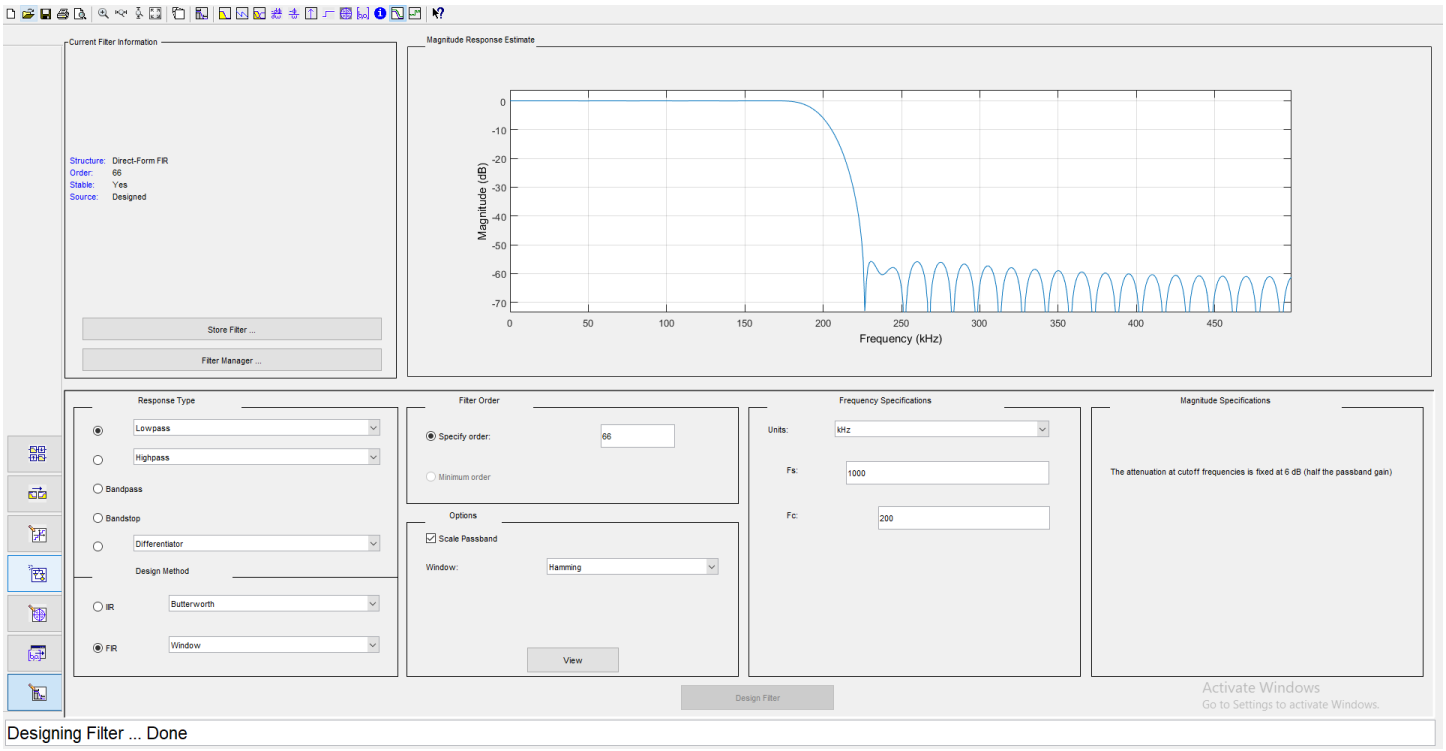
Start by opening MATLAB Filter Designer tool, by typing:

```
filterDesigner
```

in the command window, if you are using MATLAB versions that are older than 2018 use this command:

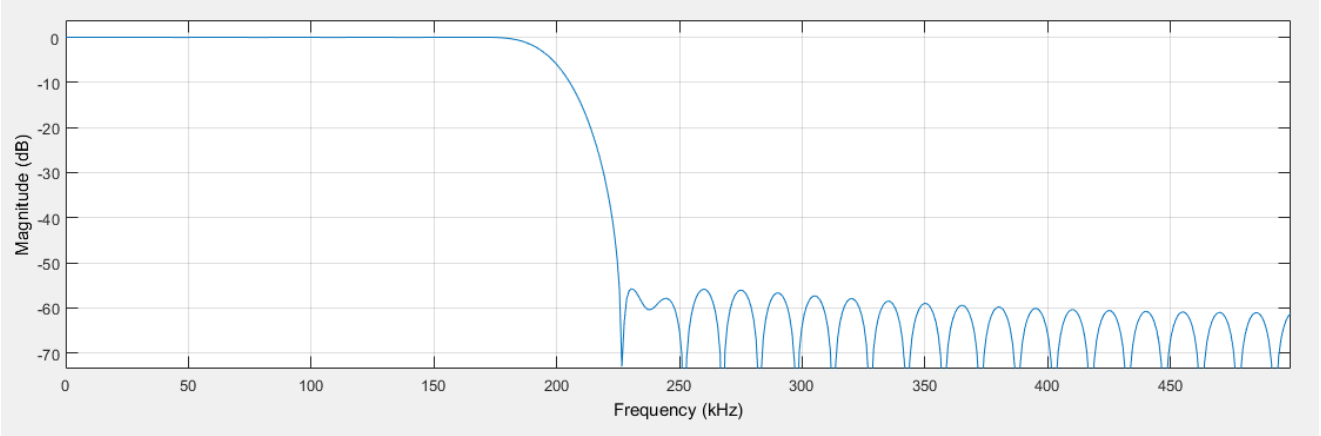
```
fdatool
```

then, choose the filter parameters that satisfies your filter specs as follows:

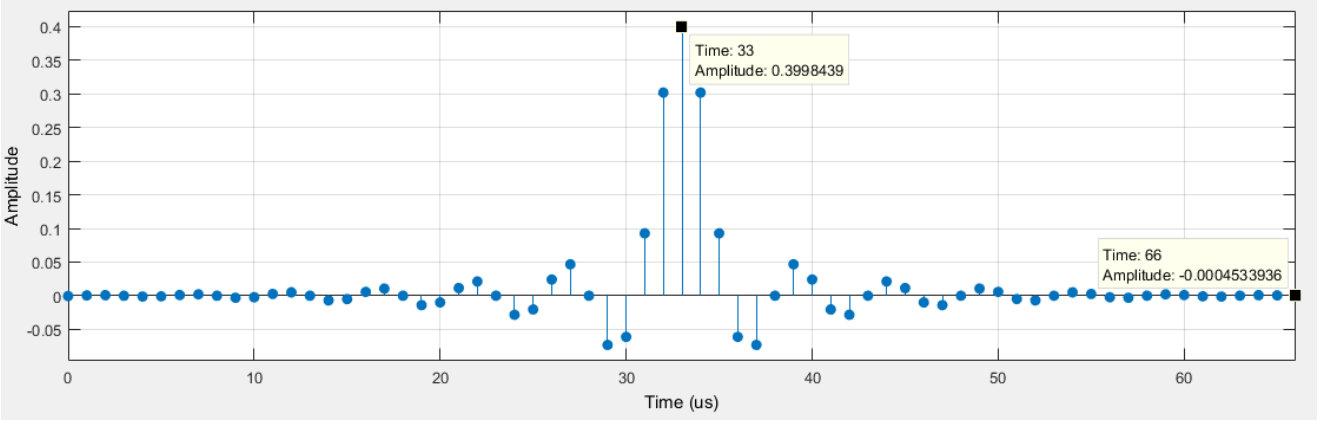


Results:

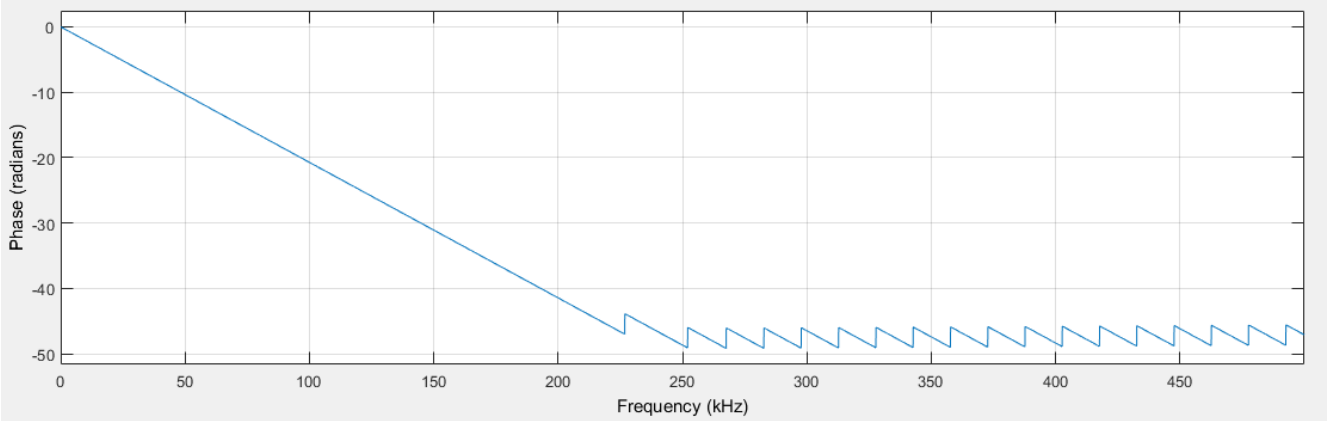
- Filter Magnitude Response



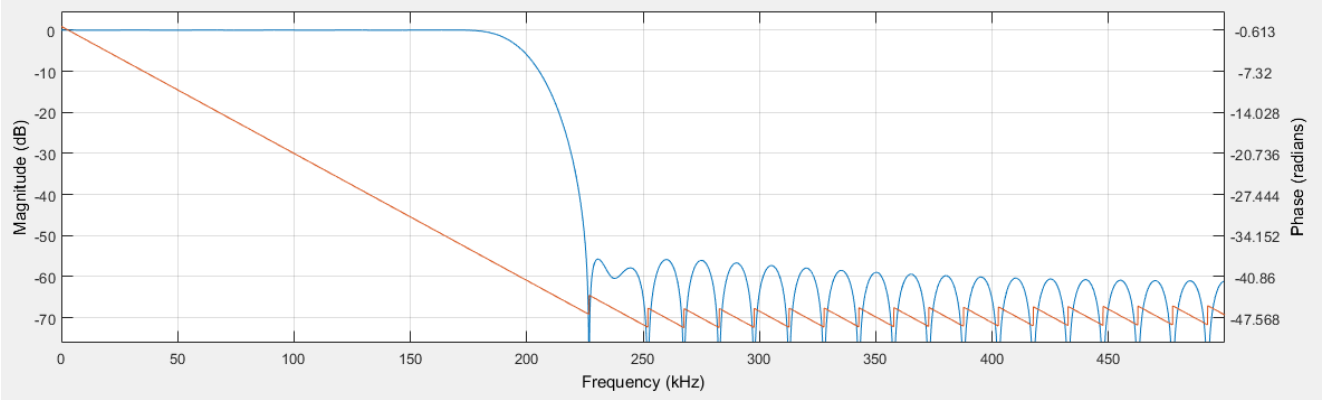
- Filter Impulse Response



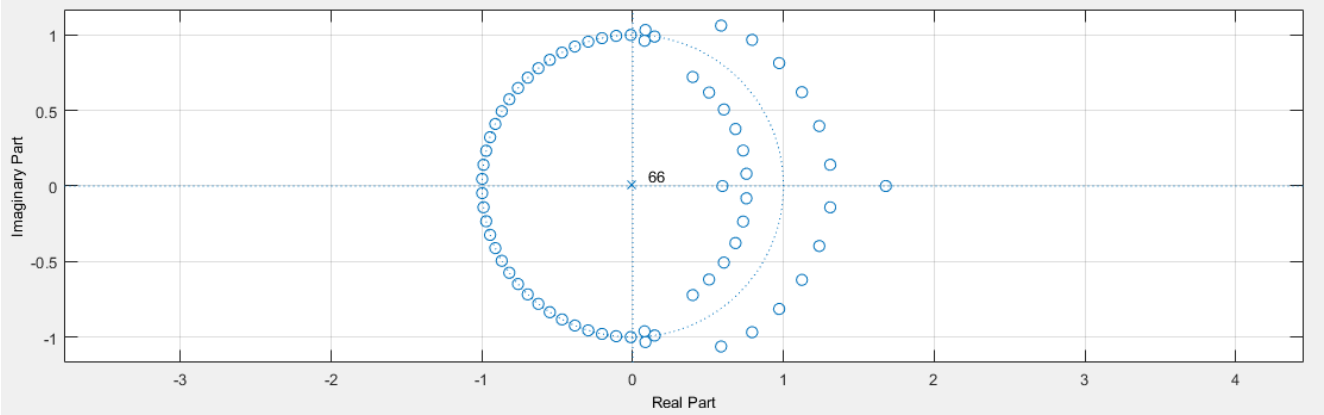
- Filter Phase Response



- Filter Magnitude VS Phase

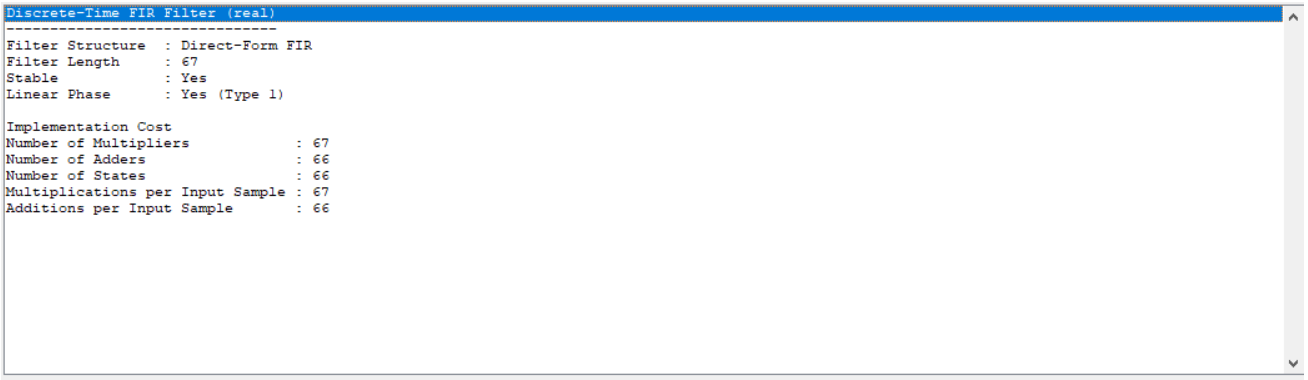


- Poles and Zeros (expecting no poles)



We can notice no poles as there is no feed-back

- Overall Filter Information



```
Discrete-Time FIR Filter (real)
-------------------------------
Filter Structure  : Direct-Form FIR
Filter Length     : 67
Stable            : Yes
Linear Phase      : Yes (Type 1)

Implementation Cost
Number of Multipliers            : 67
Number of Adders                 : 66
Number of States                 : 66
Multiplications per Input Sample : 67
Additions per Input Sample       : 66
```

- Filter coefficients

```
Numerator:
-0.00045339360612670949
 0.00047973581278097031
 0.00086208272295155151
-0.00000000000000000015373571315172554
-0.001178631675972497
-0.00087591553826649453
 0.0010599694686552923
 0.0020765691792760041
-0.00000000000000000032277104694982683
-0.0030105420816176906
-0.0022213317236098874
 0.0026353503356114946
 0.005027822982406123
-0.00000000000000000060717419041172812
-0.0068733015780619543
-0.0049305486530802729
 0.005699993246029024
 0.010627142707749416
-0.00000000000000000094371404784928613
-0.014016347219432207
-0.009933196351378553
 0.011394348015500615
 0.021182880901727742
-0.00000000000000001257568014738936 5
-0.028285397899600374
-0.020407163112107638
 0.024088386125298629
 0.046752465352924912
-0.00000000000000001478957220246358 1
-0.07316031811300637
-0.061179994949448341
 0.092735081194303931
 0.30198226783282273
 0.39984394427231884
 0.30198226783282273
 0.092735081194303931
-0.061179994949448341
-0.07316031811300637
-0.00000000000000001478957220246358 1
 0.046752465352924912
 0.024088386125298629
-0.020407163112107638
-0.028285397899600374
```

```
-0.00000000000000001257568014738936 5
 0.021182880901727742
 0.011394348015500615
-0.009933196351378553
-0.014016347219432207
-0.00000000000000000943714047849286 13
 0.010627142707749416
 0.005699993246029024
-0.0049305486530802729
-0.0068733015780619543
-0.00000000000000000607174190411728 12
0.005027822982406123
 0.0026353503356114946
-0.002221317236098874
-0.0030105420816176906
-0.00000000000000000322771046949826 83
 0.0020765691792760041
 0.0010599694686552923
-0.00087591553826649453
-0.001178631675972497
-0.00000000000000000153735713151725 54
 0.00086208272295155151
 0.00047973581278097031
-0.00045339360612670949
```

Before jumping to our next step in designing the FIR filter, we need to make these resulted filter coefficient ready to be used in a *Verilog* code, hence, we need to convert each coefficient to it's binary representation and this is done as following:

Filter Coefficients conversion to binary:

- **If the value is positive:** for example (`0.021182880901727742`)

1) First we multiply the value by 2^(the input max ragne – 1) :

   >> 0.021182880901727742 * 2^(15) = 694.120641

   (Here we let our input to be of 16-bits)

2) Then, we round the result to 694 (note, if the result is 694.8 we would also round it to the same value 694
3) Then we convert this decimal value to it's binary representation

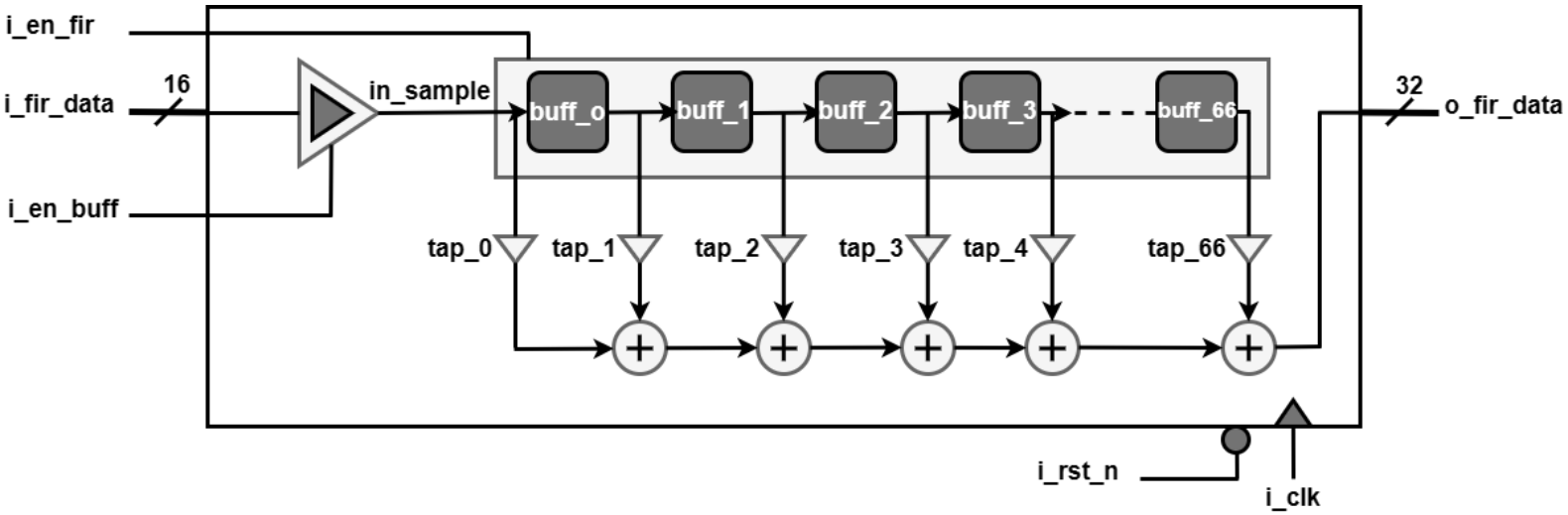
   >> 694 = (0000 0010 1011 0110)$_2$

4) For better readability , we will convert the binary value to hexadecimal representation : $(02B6)_{16}$

- **If the value is negative:** we will do the same steps except in step 3 we will get the two's complement of our negative decimal number

b) Block Diagram

We will map the main FIR Block Diagram with adding our inputs and outputs:



Here, our input will be of 16-bits , and the filter taps are of 16-bits hence, the filter output will be if 32-bits

c) RTL Design

First, let's start with I/O signals documentation:

Signal	width	Description
i_clk	1	System clock (50 MHZ for FPGAs)
i_rst_n	1	Asynchronous negative-edge reset
i_en_fir	1	Fir block enable
i_en_buff	1	Input buffer enable
i_fir_data	16	Input data (samples)
o_fir_data	32	Output filtered data

- The input and output signals will be of type **signed** as they can include negative samples.
- Taps, buffers, and accumulators will be internal regs for the fir design.

You can review the full RTL Design and Testbench on my **GitHub**: **Click Here** or scan the QR code:

d) Testbench

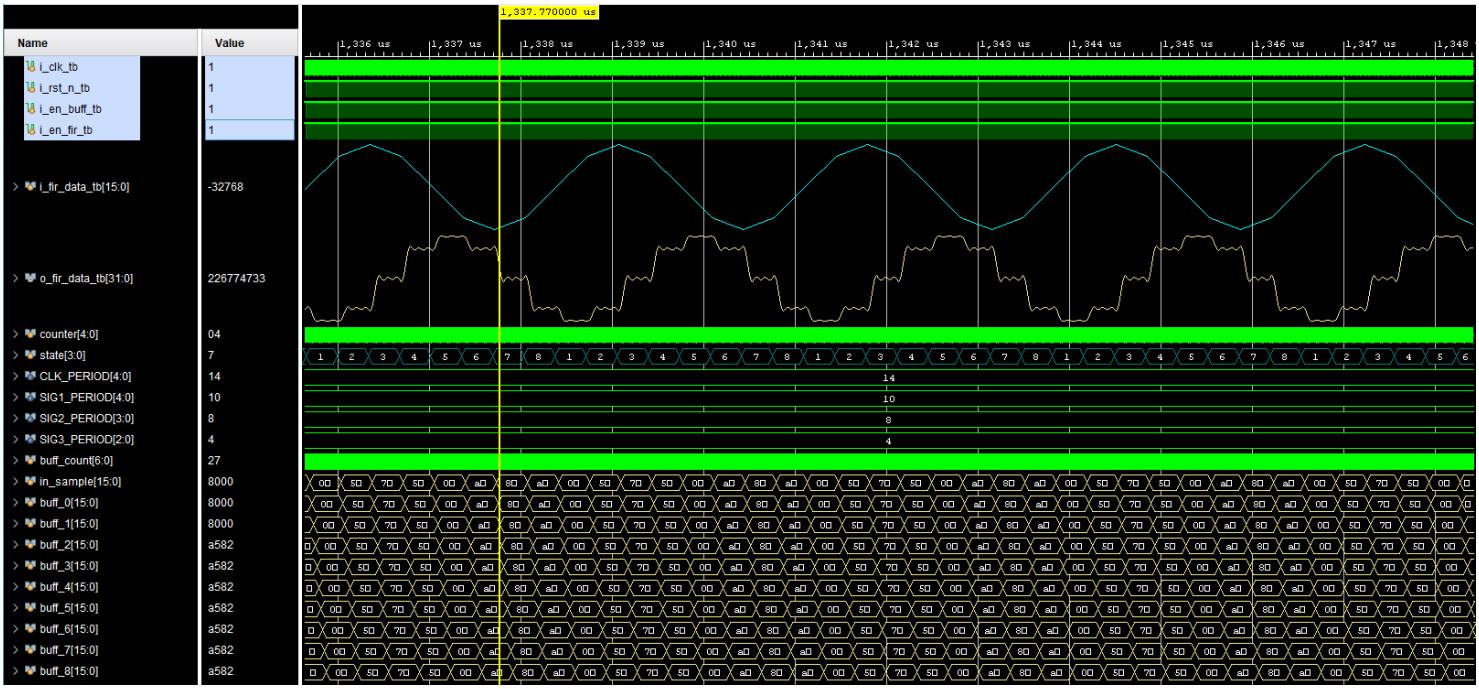
To test the FIR Filter, we will simulate an input sinusoidal signal behavior as a FSM, this signal will have 3 cases:

1. 100 KHZ : expecting it to pass with no attenuation
2. 200 KHZ : expecting the start of the attenuation effect
3. 400 KHZ : expecting a huge attenuation

(As the cutoff frequency is 200 KHZ)

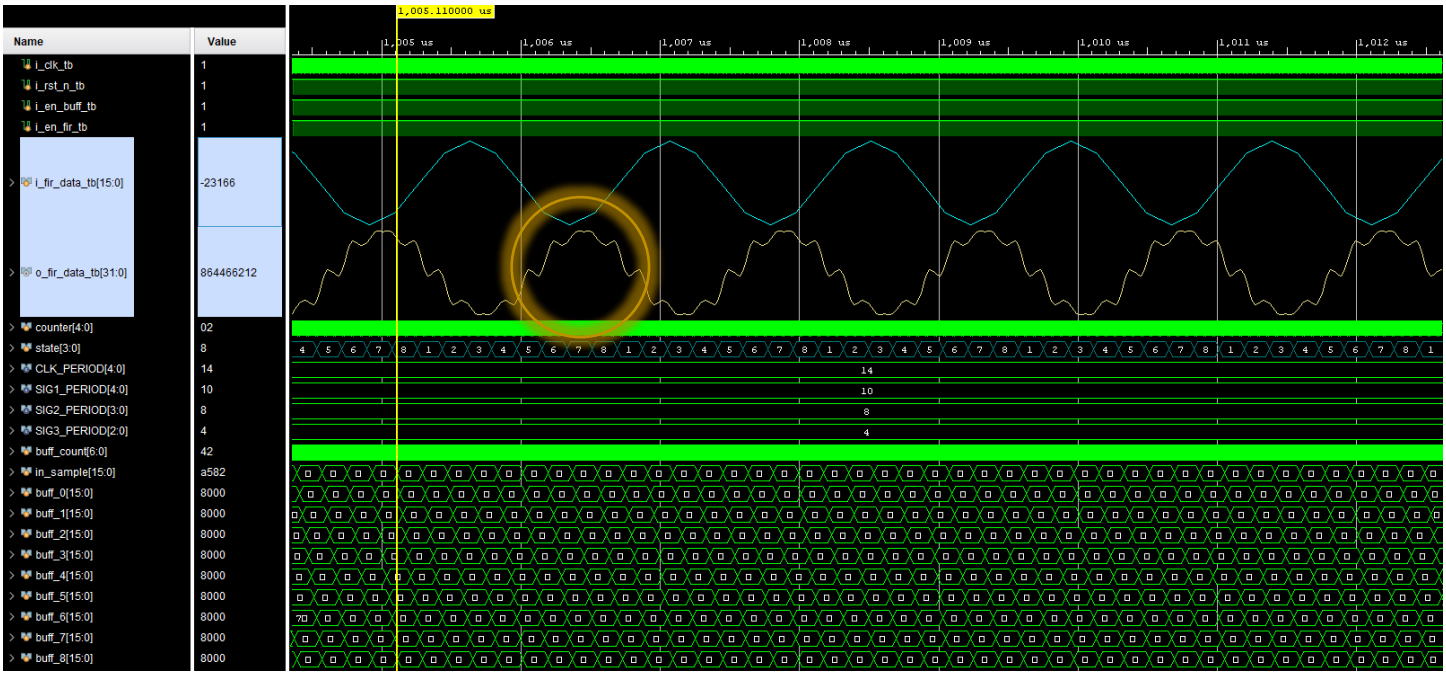
So, lets review the test results in the waveforms:

1. 100 KHZ



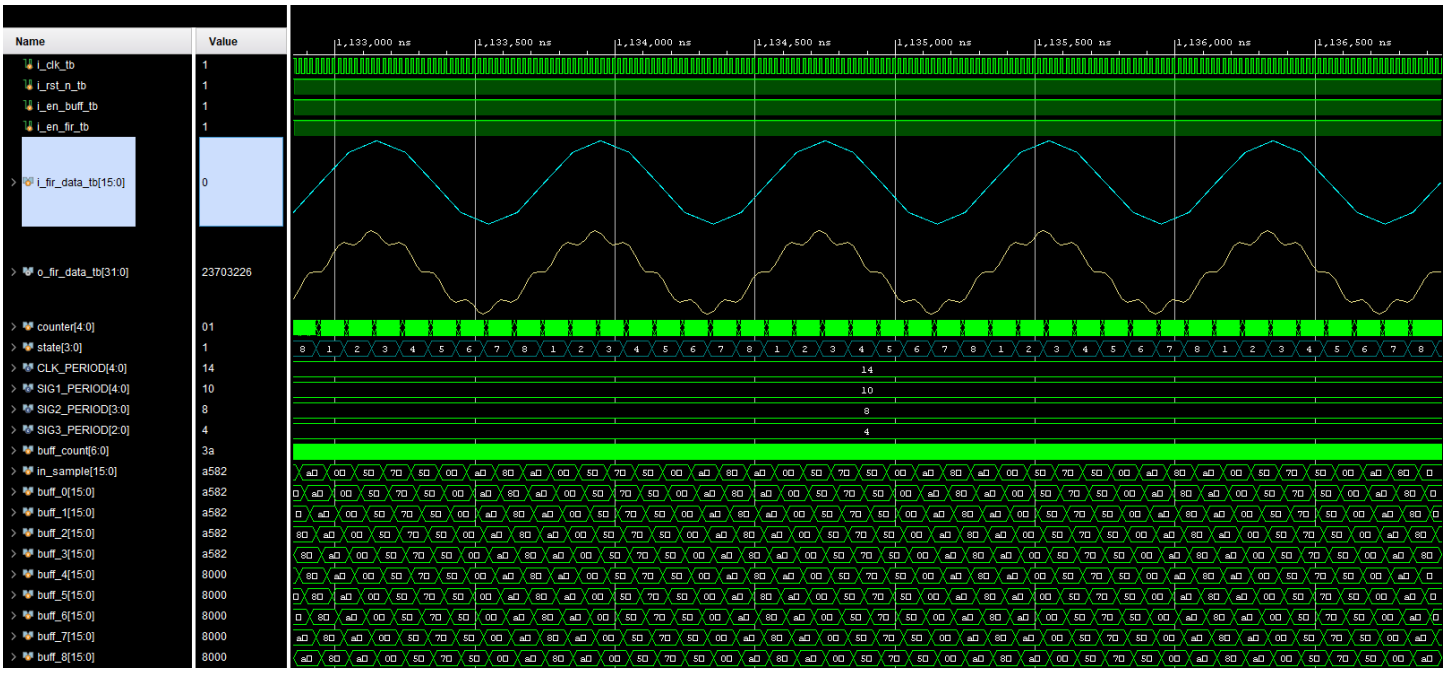
We can notice that the filtered signal follows the input signal with some ripples that is a very normal behavior of the FIR filters output, and it's also delayed of the input due to the filter shifting behavior in the buffers.

2. 200 KHZ



We can notice the start of the attenuating effect as the input signal is 200 KHZ and the cutoff frequency of the filter is 200 KHZ.

3. 400 KHZ



We can notice the huge attenuation effect on the output signal as the input signal is 400 KHZ which exceeds the filer cutoff frequency.

LinkedIn: Nour El-Deen Samir