

Johns Hopkins
Engineering for Professionals
525.742 SoC FPGA Design Lab

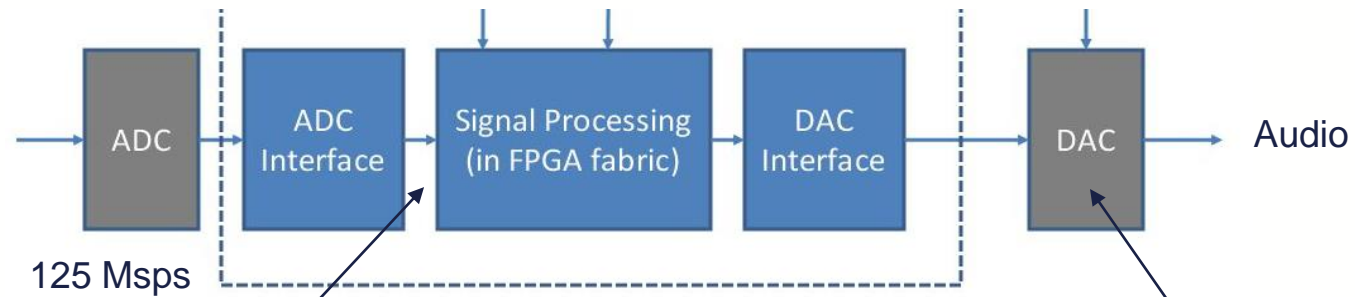
Lecture 4 - Filtering

Agenda

- Lab 4 Topics
 - Channel Selection
 - Finite Impulse Response Filters
 - *Design (Matlab)*
 - *FPGA Implementation*
 - Computational requirements
 - Fixed point vs floating point issues
 - Modelling of FPGA filter in Matlab
- Demonstrate Lab 4



In Progress : Our Radio Receiver Core



125 Million 16-bit numbers / sec

~48k 16-bit samples (L+R)/ second

Lab 4 : Filter

For lab 4, we will presume that our signal of interest is located at the frequencies 0-18kHz, we will design a suitable filter, simulate it and demonstrate it using the framework we started in Lab 4

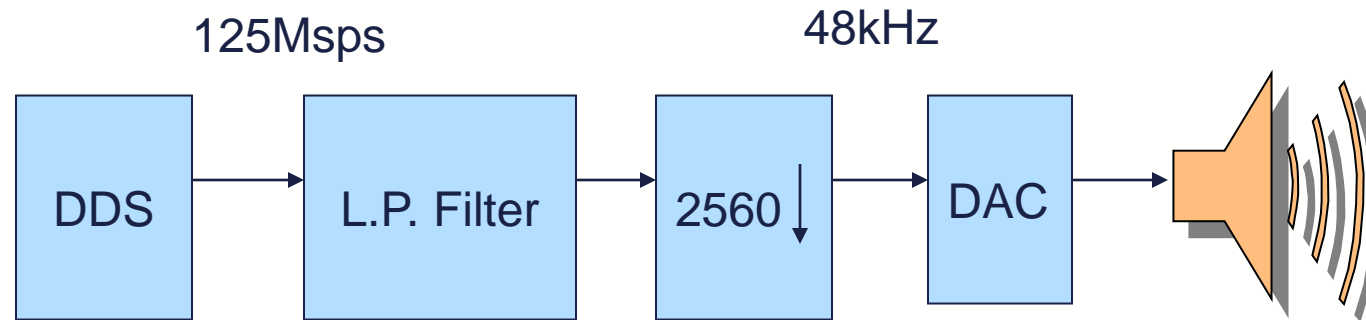


■ Filter Task

- Keep 0-18kHz relatively unchanged
- Remove (suitably attenuate) all other frequencies from the signal in preparation for reducing the sample rate
 - *New sample rate will be approx 48.8kHz*

Lab 4 : Filter Demonstration

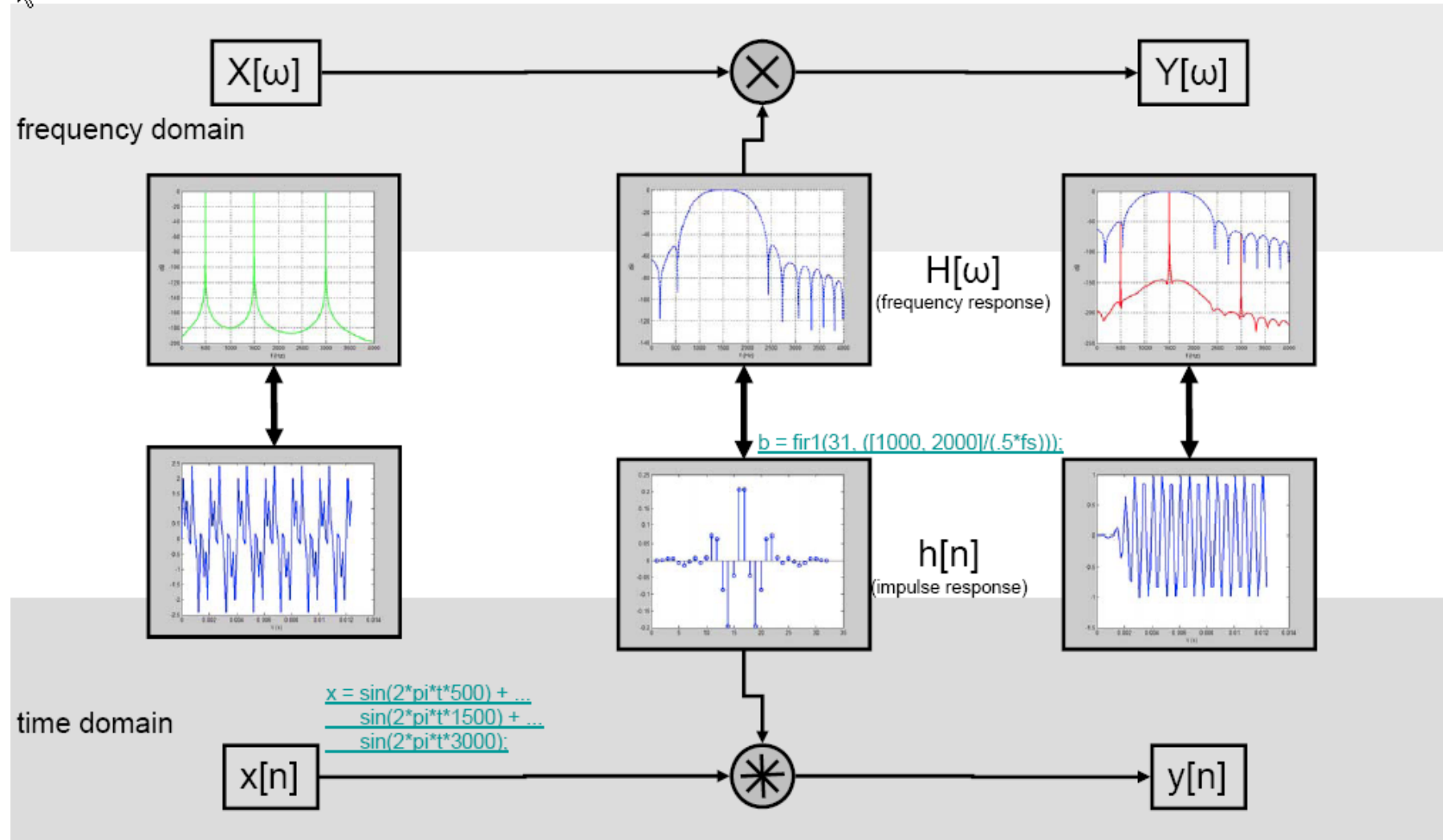
- Implement the channel selection filter which will be used in the SDR.



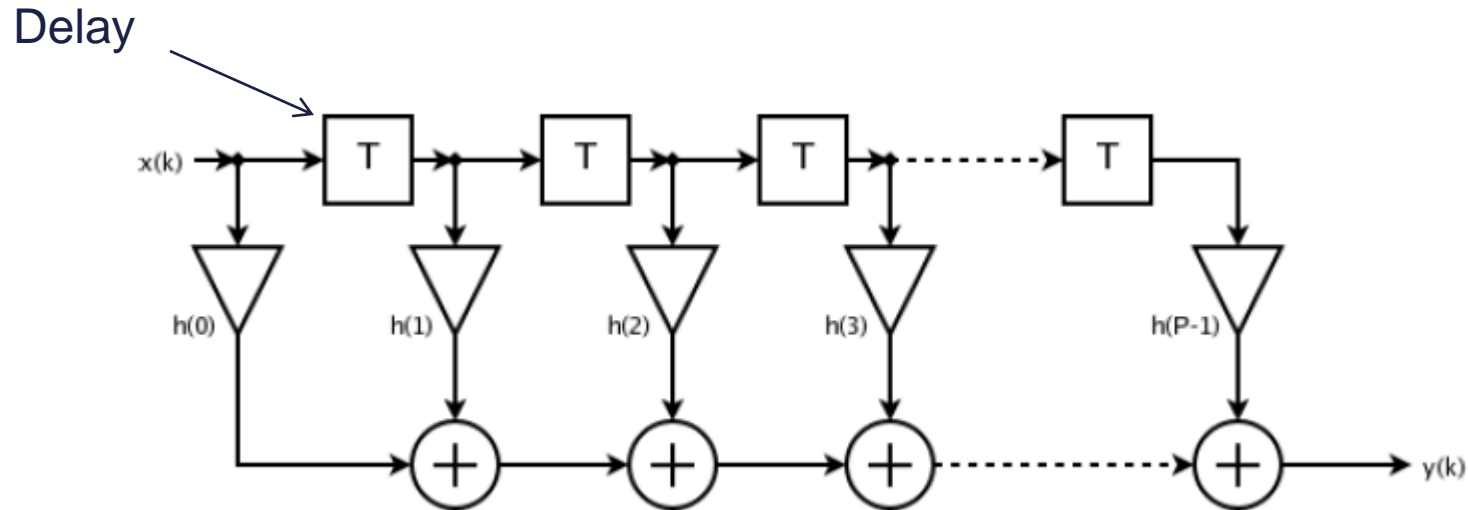
the DDS is a convenient simulator for signals that will eventually be received from an A/D converter

Lab 4 is basically **exactly** like Lab 3, except without the aliasing!

FIR Filtering



FIR Implementation



- Convolution in time domain is computationally taxing, yet fairly straightforward.

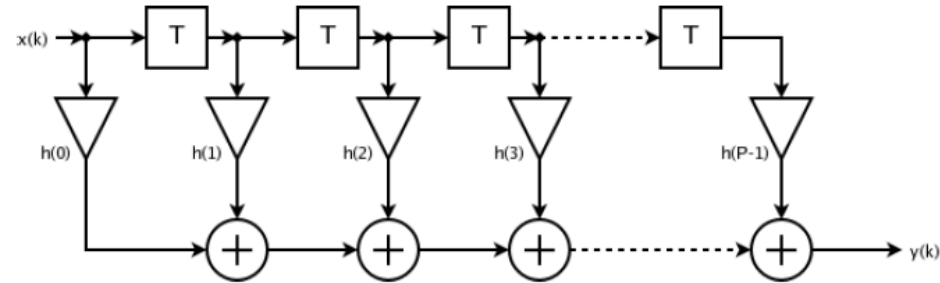
$$\begin{aligned}(f * g)[n] &\stackrel{\text{def}}{=} \sum_{m=-\infty}^{\infty} f[m] g[n - m] \\ &= \sum_{m=-\infty}^{\infty} f[n - m] g[m]. \quad (\text{commutativity})\end{aligned}$$

Example Convolution

Filter length (N) = 3

$h = [.2 \ .6 \ .2]$

Input data = $[0 \ 20 \ -1 \ -20 \ 0 \ 20 \ \dots]$



```
>> h = [.2 .6 .2];
```

```
>> x = [ 0 20 -1 -20 0 20];
```

```
>> y = filter(h,[1],x) % [1] for FIR filter
```



```
[ 0  4.0000  11.8000 -0.6000 -12.2000  0]
```

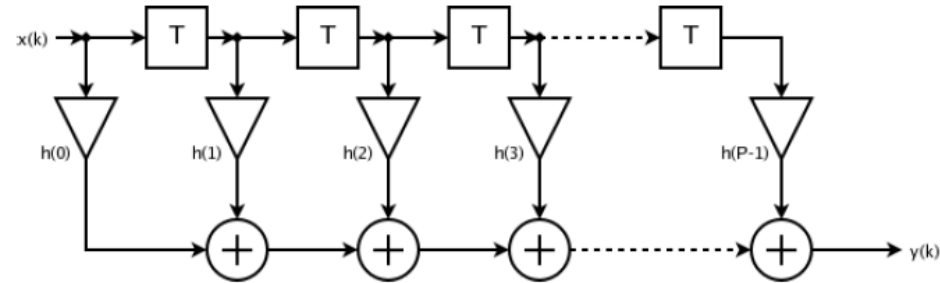
$$\begin{aligned} y[0] &= x[0] * h[0] = .2 * 0 = 0 \\ y[1] &= x[1] * h[0] + x[0] * h[1] = 20 * .2 \\ y[2] &= x[2] * h[0] + x[1] * h[1] + x[0] * h[2] = \\ y[3] &= x[3] * h[0] + x[2] * h[1] + x[1] * h[2] = \\ y[4] &= x[4] * h[0] + x[3] * h[1] + x[2] * h[2] = \\ &\dots \end{aligned}$$

Example Convolution

Filter length (N) = 3

$h = [.2 \ .6 \ .2]$

Input data = [20 20 20 20 20 20 ...]



```
>> h = [.2 .6 .2];
```

```
>> x = [ 20 20 20 20 20 20 20];
```

```
>> y = filter(h,[1],x) % [1] for FIR filter
```



[4 16 20 20 20 20]

$$y[0] = x[0] * h[0] = 4$$

$$y[1] = x[1] * h[0] + x[0] * h[1] = 16$$

$$y[2] = x[2]h[0] + x[1]h[1] + x[0]h[2] = 20$$

$$y[3] = x[3]h[0] + x[2]h[1] + x[1]h[2] = 20$$

$$y[4] = x[4]h[0] + x[3]h[1] + x[2]h[2] = 20$$

....

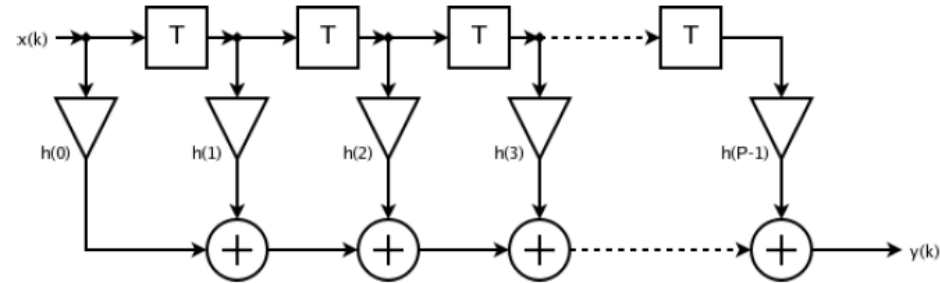


Example Convolution

Filter length (N) = 3

$h = [.2 \ .6 \ .2]$

Input data = $[0 \ 0 \ 1 \ 0 \ 0]$



```
>> h = [.2 .6 .2];
```

```
>> x = [ 0 0 1 0 0];
```

```
>> y = filter(h,[1],x) % [1] for FIR filter
```



$[0 \ 0 \ .2 \ .6 \ .2]$

$$y[0] = x[0] * h[0] = 0$$

$$y[1] = x[1] * h[0] + x[0] * h[1] = 0$$

$$y[2] = x[2]h[0] + x[1]h[1] + x[0]h[2] = .2$$

$$y[3] = x[3]h[0] + x[2]h[1] + x[1]h[2] = .6$$

$$y[4] = x[4]h[0] + x[3]h[1] + x[2]h[2] = .2$$

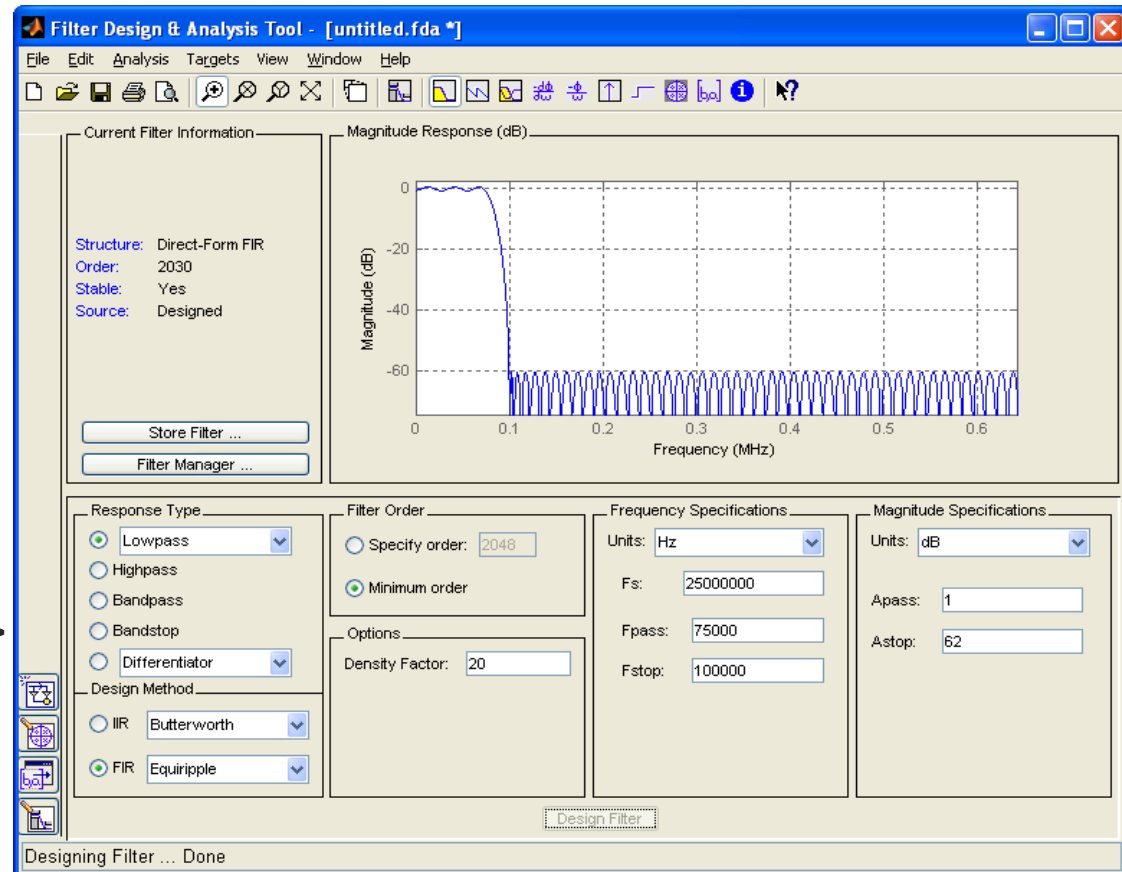
....



Example Filter Design Process

Please note that these are not the requirements for Lab 4!, this is an exploration of the design process and case study for design criteria

- Filter Specs:
 - $F_s = 25 \text{ Msp}$
 - 75kHz passband
 - 100k stopband
 - Attenuate stopband signals by > 60dB
- FIR Filter with <2048 coefficients will meet this requirement



Matlab : "fdatool" (goes from specs to $h[n]$)

Instructor provided code will put these in a special Xilinx format

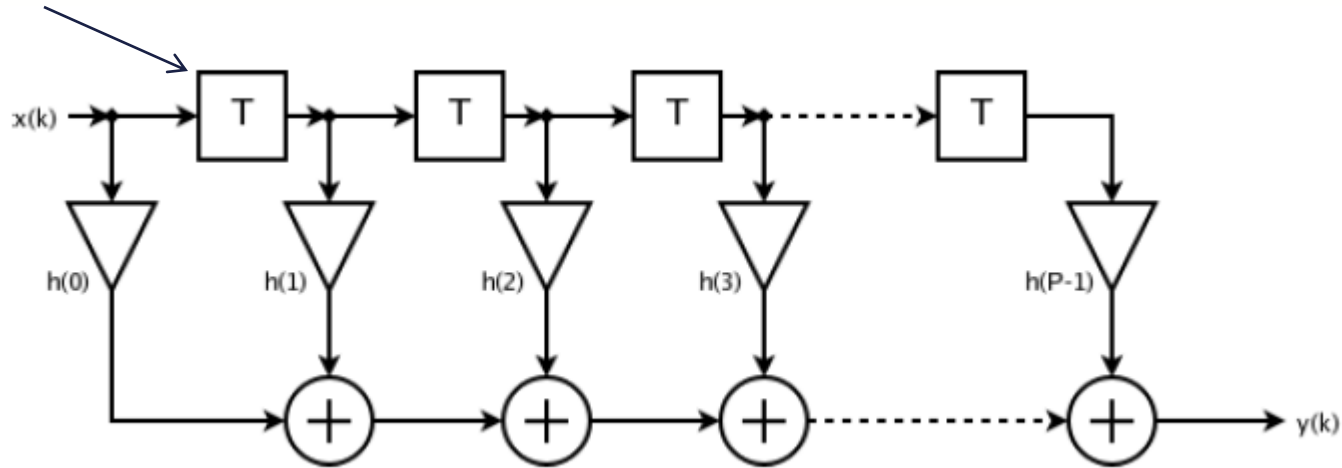
Questions

- Computational requirements :
 - Can our FPGA do the required computation?
 - How much space will it take?
Multipliers, slices, ***RAM***?
 - How fast must the clock be?
- Numerical accuracy
 - What will be the accuracy differences in our FPGA implementation due to fixed point issues?



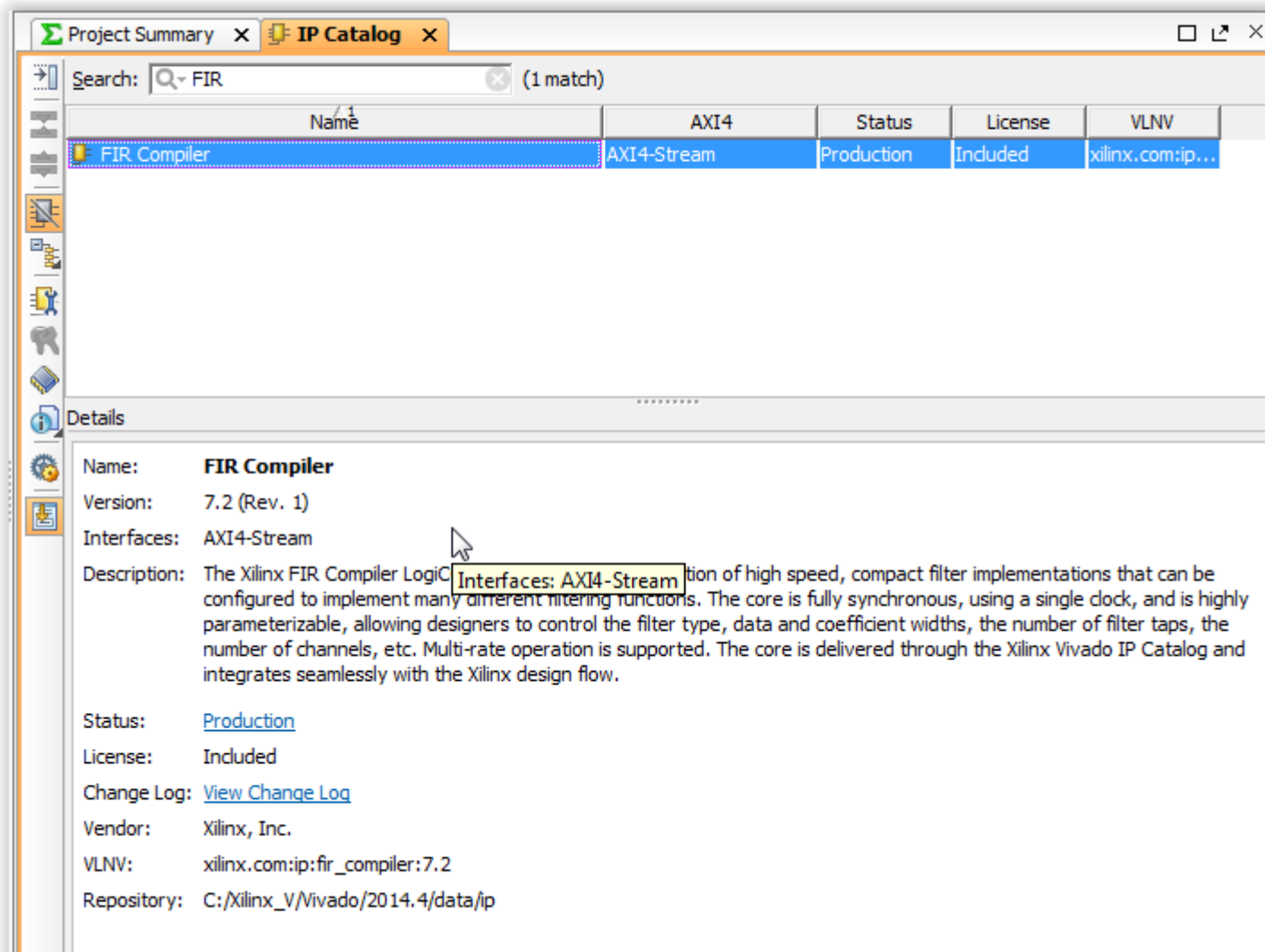
FIR Implementation

Delay



- How many Multiply-Accumulate operations (MAC) are required per sample to implement the filter we designed?
- What is the overall rate of MACs per second?

FIR Compiler



The screenshot displays the Xilinx Vivado IP Catalog window. The search bar at the top shows "FIR" with "(1 match)". Below the search bar is a table listing the IP core. The table has columns for Name, AXI4, Status, License, and VLN. The "FIR Compiler" is listed with AXI4-Stream interface, Production status, Included license, and VLN xilinx.com:ip:7.2. Below the table is a "Details" section for the "FIR Compiler".

Name	AXI4	Status	License	VLN
FIR Compiler	AXI4-Stream	Production	Included	xilinx.com:ip:7.2

Details

Name: **FIR Compiler**

Version: 7.2 (Rev. 1)

Interfaces: AXI4-Stream

Description: The Xilinx FIR Compiler Logic is a high speed, compact filter implementation that can be configured to implement many different filtering functions. The core is fully synchronous, using a single clock, and is highly parameterizable, allowing designers to control the filter type, data and coefficient widths, the number of filter taps, the number of channels, etc. Multi-rate operation is supported. The core is delivered through the Xilinx Vivado IP Catalog and integrates seamlessly with the Xilinx design flow.

Status: [Production](#)

License: Included

Change Log: [View Change Log](#)

Vendor: Xilinx, Inc.

VLN: xilinx.com:ip:fir_compiler:7.2

Repository: C:/Xilinx_V/Vivado/2014.4/data/ip

Familiar AXI-Stream interface for in/out

The screenshot displays the Xilinx IP Integrator interface for the FIR Compiler IP. The left pane shows the IP Symbol with the following ports: S_AXIS_DATA (input), M_AXIS_DATA (output), and ack (output). The right pane shows the Filter Options tab with the following configuration:

- Component Name: fir_compiler_0
- Filter Options: Channel Specification, Implementation, Detailed Implementation, Interface, Summary
- Filter Coefficients:
 - Select Source: COE File
 - Coefficient Vector: 6,0,-4,-3,5,6,-6,-13,7,44,64,44,7,-13,-6,6,5,-3,-4,0,6
 - Coefficient File: venstds1/Documents/MATLAB/fir_inclass/filt2030_float.coe
 - Number of Coefficient Sets: 1 [1 - 1024]
 - Number of Coefficients (per set): 2031
 - Use Reloadable Coefficients: ☐
- Filter Specification:
 - Filter Type: Single Rate
 - Inferred Coefficient Structure(s): Symmetric or Non Symmetric
 - Rate Change Type: Integer
 - Interpolation Rate Value: 1 [1 - 1]
 - Decimation Rate Value: 1 [1 - 1]
 - Zero Pack Factor: 1 [1 - 1]

Tool takes as input the “COE” file (which is just a list of coefficients) to get $h[n]$

IP Symbol

Freq. Response

Implementation Details

Resource Estimates

DSP slice count:

1016

BRAM count:

0

Information

Start-up Latency:

1039

Calculated Coefficients:

2031

Coefficient front padding:

0

Clock cycles per input:

1

Clock cycles per output:

1

Processing cycles per output:

1

AXI4 Stream Port Structure

S_AXIS_DATA - TDATA

Transaction	Field	Type
0	REAL(15:0)	fix16_0

M_AXIS_DATA - TDATA

Transaction	Field	Type
0	REAL(38:0)	fix39_22

Interleaved Channel Sequences

Component Name

fir_compiler_0

Filter Options

Channel Specification

Implementation

Detailed Implementation

Interface

Summary

Interleaved Channel Specification

Channel Sequence

Basic

Number of Channels

1

[1 - 1024]

Select Sequence

All

Sequence ID List

P4-0,P4-1,P4-2,P4-3,P4-4

Parallel Channel Specification

Number of Paths

1

[1 - 16]

Hardware Oversampling Specification

Select Format

Frequency Specification

Sample Period (Clock Cycles)

1

[1.0 - 1.0E7]

Input Sampling Frequency (MHz)

25.0

[1.0E-6 - 47488.0]

Clock Frequency (MHz)

25

[0.390625 - 742.0]

Implementation details show what we computed previously with a savings
Due to symmetrical structure

DSP in Xilinx FPGAs

2016 V6 DSP Slices @ 600MHz fmax

FPGA Comparison Table

Features	Artix-7	Kintex-7	Virtex-7	Spartan-6	Virtex-6
Logic Cells	215,000	480,000	2,000,000	150,000	760,000
BlockRAM	13Mb	34Mb	68Mb	4.8Mb	38Mb
DSP Slices	740	1,920	3,600	180	2,016
DSP Performance (symmetric FIR)	930GMACs	2,845GMACs	5,335GMACs	140GMACs	2,419GMACs

180 S6 DSP Slices @ 390MHz fmax

Zynq 7Z020 : 220 DSP Slices, 276GMACs



IP Symbol Freq. Response **Implementation Details** ◀ ▶ ☰

Component Name

Filter Options **Channel Specification** Implementation Detailed Implementation Interface Summary

Resource Estimates

DSP slice count:	52
BRAM count:	0

Information

Start-up Latency:	79
Calculated Coefficients:	2039
Coefficient front padding:	4
Clock cycles per input:	20
Clock cycles per output:	20
Processing cycles per output:	20

AXI4 Stream Port Structure

S_AXIS_DATA - TDATA

Transaction	Field	Type
0	REAL(15:0)	fix16_0

M_AXIS_DATA - TDATA

Transaction	Field	Type
0	REAL(38:0)	fix39_22

Interleaved Channel Specification

Channel Sequence

Number of Channels [1 - 1024]

Select Sequence

Sequence ID List

Parallel Channel Specification

Number of Paths [1 - 16]

Hardware Oversampling Specification

Select Format

Sample Period (Clock Cycles) [1.0 - 1.0E7]

Input Sampling Frequency (MHz) [1.0E-6 - 47488.0]

Clock Frequency (MHz) [0.390625 - 742.0]

Interleaved Channel Sequences

This ratio is a promise from you to the tools!, Don't violate it!

If we let filter run at higher clock rate than input sample rate, structure is automatically adapted such that a convolution takes multiple clocks using Shared multipliers

Zynq DSP Slice (>600MHz)

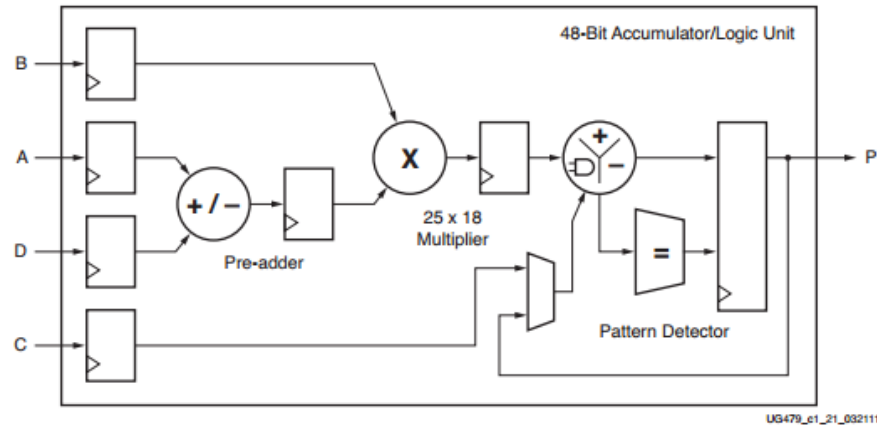


Figure 1-1: Basic DSP48E1 Slice Functionality

Some highlights of the DSP functionality include:

- 25 × 18 two's-complement multiplier:
 - Dynamic bypass
- 48-bit accumulator:
 - Can be used as a synchronous up/down counter
- Power saving pre-adder:
 - Optimizes symmetrical filter applications and reduces DSP slice requirements

- $220 * 600M > 130GMACs$
 - Our FPGA is capable of doing what we want
 - There will be other features of the FPGA that need multipliers...
- These figures should help put FPGA capabilities for DSP in perspective

IP Symbol Freq. Response **Implementation Details** ◀ ▶ ☰



Component Name `fir_compiler_0`

Filter Options Channel Specification Implementation Detailed Implementation Interface Summary

Filter Coefficient **Filter Options**

Select Source `COE File`

Coefficient Vector `6,0,-4,-3,5,6,-6,-13,7,44,64,44,7,-13,-6,6,5,-3,-4,0,6`

Coefficient File `venstds1/Documents/MATLAB/fir_inclass/filt2030_float.coe`  

Number of Coefficient Sets `1` [1 - 1024]

Number of Coefficients (per set): 2031

☐ Use Reloadable Coefficients

Filter Specification

Filter Type `Decimation`

Inferred Coefficient Structure(s) : Symmetric or Non Symmetric

Rate Change Type `Integer`

Interpolation Rate Value `1` [1 - 1]

Decimation Rate Value `128` [2 - 1024]

Zero Pack Factor `1` [1 - 1]

Resource Estimates

DSP slice count: 9

BRAM count: 1

Information

Start-up Latency: 144

Calculated Coefficients: 2047

Coefficient front padding: 8

Clock cycles per input: 1

Clock cycles per output: 128

Processing cycles per output: 1

AXI4 Stream Port Structure

S_AXIS_DATA - TDATA

Transaction	Field	Type
0	REAL(15:0)	fix16_0

M_AXIS_DATA - TDATA

Transaction	Field	Type
0	REAL(38:0)	fix39_22

Interleaved Channel Sequences

Allowing the filter to use all the time between *output* samples for the convolution makes the task easily achievable (this is back to the 25MHz clock!)

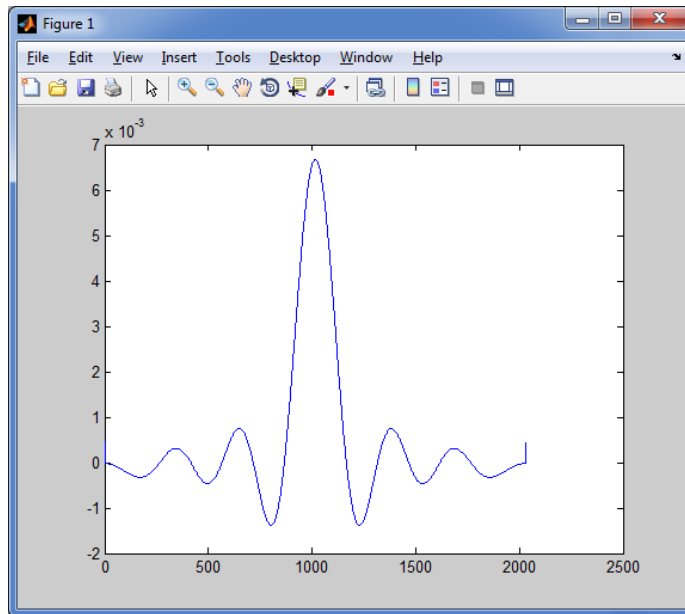
Questions

- Computational requirements :
 - Can our FPGA do the required computation?
 - How much space will it take?
 - ***Multipliers, slices, RAM?***
 - How fast must the clock be?
- Numerical accuracy
 - What will be the accuracy differences in our FPGA implementation due to fixed point issues?




Scaling and Quantizing $h[n]$

Output from FDATool is floating point coefficients $h[\text{NUMTAPS}]$ (aka “impulse response”). In absense of floating point multipliers, this is not directly implementable on our FPGA. We need to transform this impulse response to integers while preserving the function.



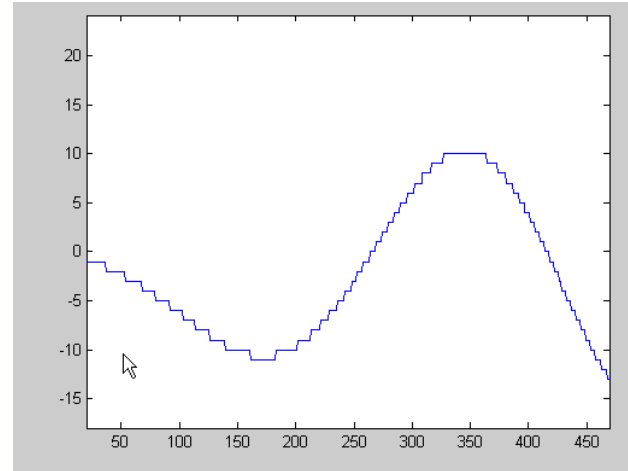
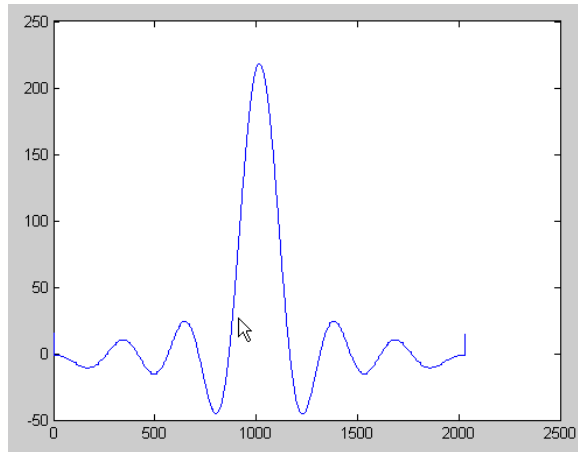
```
Numint = int32(Num*ScaleFactor+0.5);
```

Scaling factor,
How to choose?

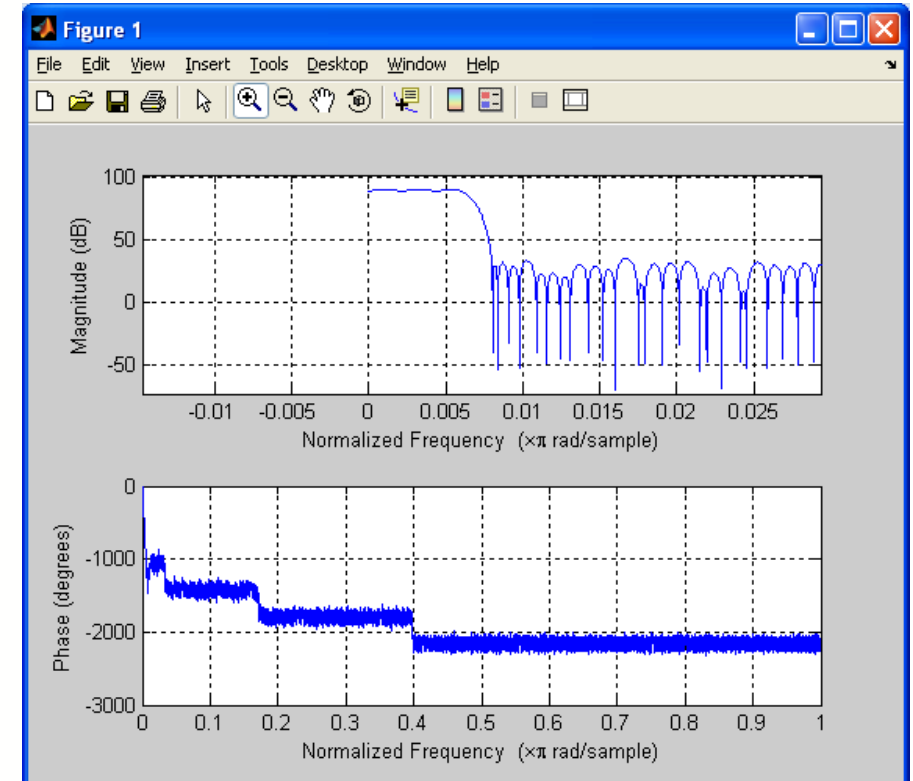


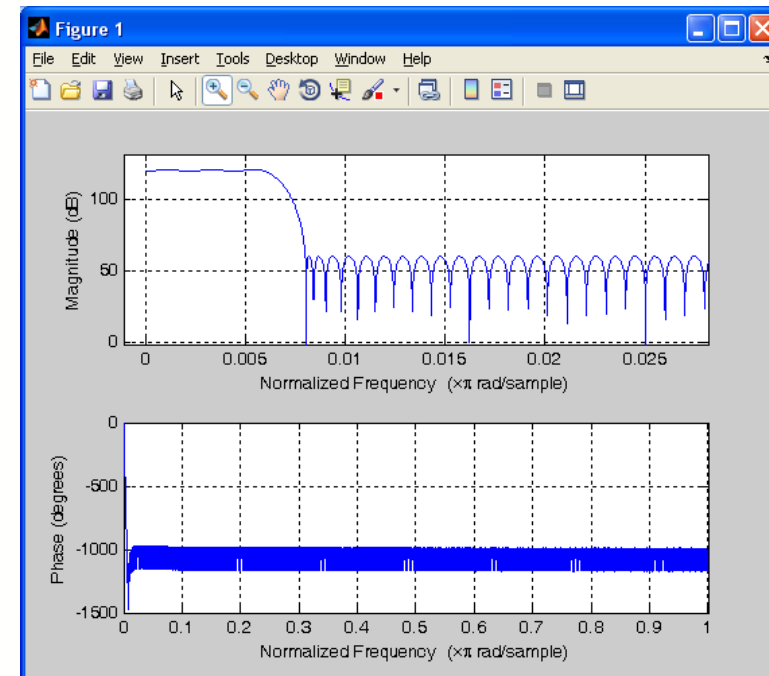
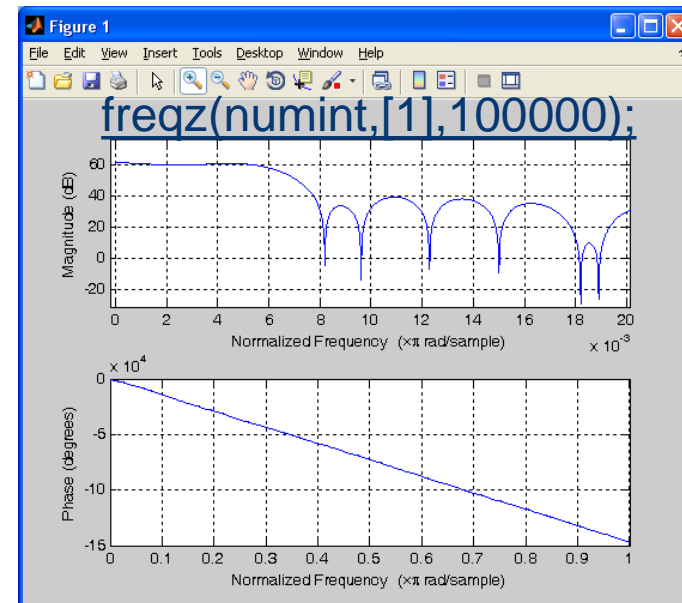
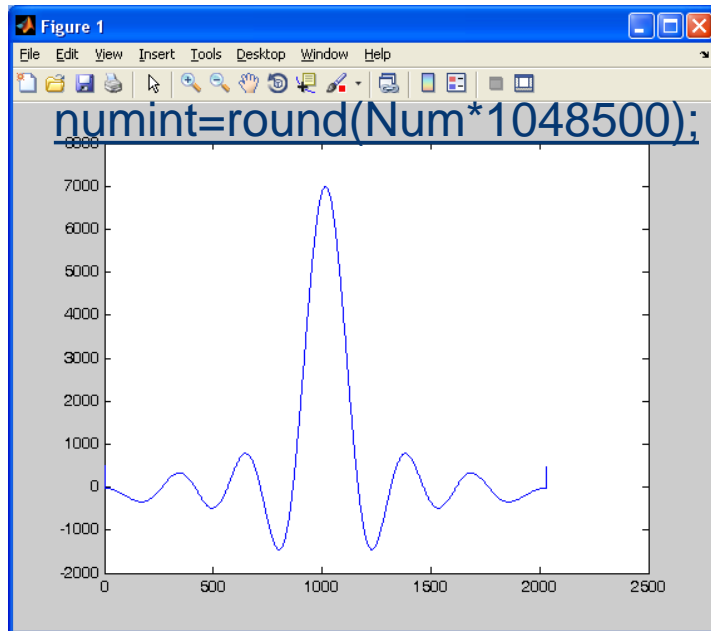
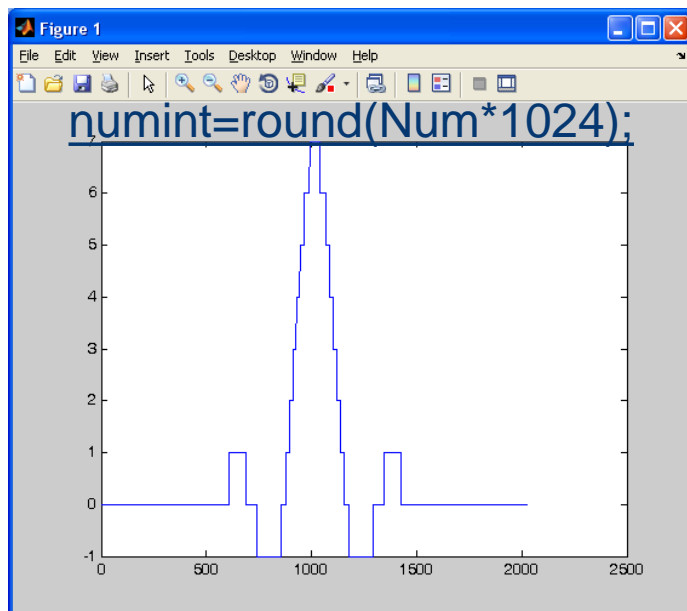
“export” the coefficients to the workspace as “Num”

Quantized Coefficients



Overall, making the coefficients integers, (after multiplying by 32700) doesn't affect our response too badly. With this scaling factor, our coefficient width is really only 9 bits. Some optimization between coefficient width and filter order could be undertaken if we chose.





Managing Coefficient Scaling

- As we have discussed, the scaling that we apply to the coefficients will affect:
 - Precision of coefficients
 - *Frequency response characteristics*
 - Filter gain
- There are two methods of handling the transition from Matlab -> Vivado
 - **Manually : we explicitly scale coefficients in matlab**
 - IP customizer driven



Scaling and Quantizing $h[n]$

```
NumInt = int32(Num*32700+0.5);  
fid=fopen('filt2030.coe','w');  
fprintf(fid,'radix=10;\ncoefdata= \n');  
fprintf(fid,'%d,\n',NumInt);  
fclose(fid);
```

Scaling factor,
Choose this for a reasonable
Performance vs. utilization tradeoff

```
radix=10;  
coefdata=  
15,  
-1,  
-1,  
-1,  
-1,  
...etc
```

Result is : impulse response of filter in a “coe” file, which we will use later when designing the filter. Note that filter gain has changed though over the unity gain filter we designed in Matlab. ***Now, signals in the passband will come out x32700 over the input level.***



Scaling and Quantizing $h[n]$

The screenshot shows the FIR Compiler GUI with the following sections:

- Documents View**
 - Implementation Details
 - Resource Estimates
 - MULT/DSP slice count: 3
 - BRAM count : 3
 - Implementation Details
 - Start-up Latency: 530
 - Calculated Coefficients: 2079
 - Coefficient front padding: 24
 - AXI4-Stream Port Structure
 - s_axis_data_tdata
 - data : 15 downto 0
 - m_axis_data_tdata
 - data : 31 downto 0
 - Interleaved Channel Sequences
- FIR Compiler** (xilinx.com:ip:fir_compiler:6.3)
 - Coefficient Options**
 - Coefficient Type : Signed
 - Quantization : Integer Coefficients
 - Coefficient Width : 9 Range: 9..35
 - ☐ Best Precision Fraction Length
 - Coefficient Fractional Bits : 0 Range: 0..0
 - Coefficient Structure : Inferred
 - Datapath Options**
 - Input Data Type : Signed
 - Input Data Width : 16 Range: 2..33
 - Input Data Fractional Bits : 0 Range: 0..16
 - Output Rounding Mode : Full Precision
 - Output Width : 32 Range: 1..32
 - Output Fractional Bits : 0

At the bottom, there are tabs for IP Sym..., Freq. Resp..., Implementation De..., and Coefficient Rel..., along with buttons for Datasheet, < Back, Page 3 of 6, Next >, Generate, Cancel, and Help.

Scaling and Quantizing $h[n]$

- Result is very simple – the pure result of convolving our new scaled $h[n]$ with the data.
 - Output will simply be scale_factor vs. the unity gain filter.
If we don't want to keep the huge number of bits, how do we reduce this number?
- Some divisions can be done with shifting (aka picking different bits)



Managing Coefficient Scaling

- As we have discussed, the scaling that we apply to the coefficients will affect:
 - Precision of coefficients
 - *Frequency response characteristics*
 - Filter gain
- There are two methods of handling the transition from Matlab -> Coregen
 - Manually : we explicitly scale coefficients in matlab
 - Vivado / GUI driven



Writing $h[n]$ to a file

```
>> fid=fopen('filt2030_float.coe','w');  
>> fprintf(fid,'radix=10;\ncoefdata=\n');  
>> fprintf(fid,'%5.15f,\n',Num);  
>> fclose(fid);
```

```
radix=10;  
coefdata=  
0.000469937006051,  
-0.000023959423709,  
-0.000023496101200,  
-0.000023104860244,  
-0.000022826569752,  
-0.000022612595620,  
-0.000022504297004,  
-0.000022452966158,  
..etc.
```

Result is : impulse response of filter in a “coe” file, which we will use later when designing the filter. Note that scaling / quantization of coefficients has not happened yet – this is the raw matlab filter



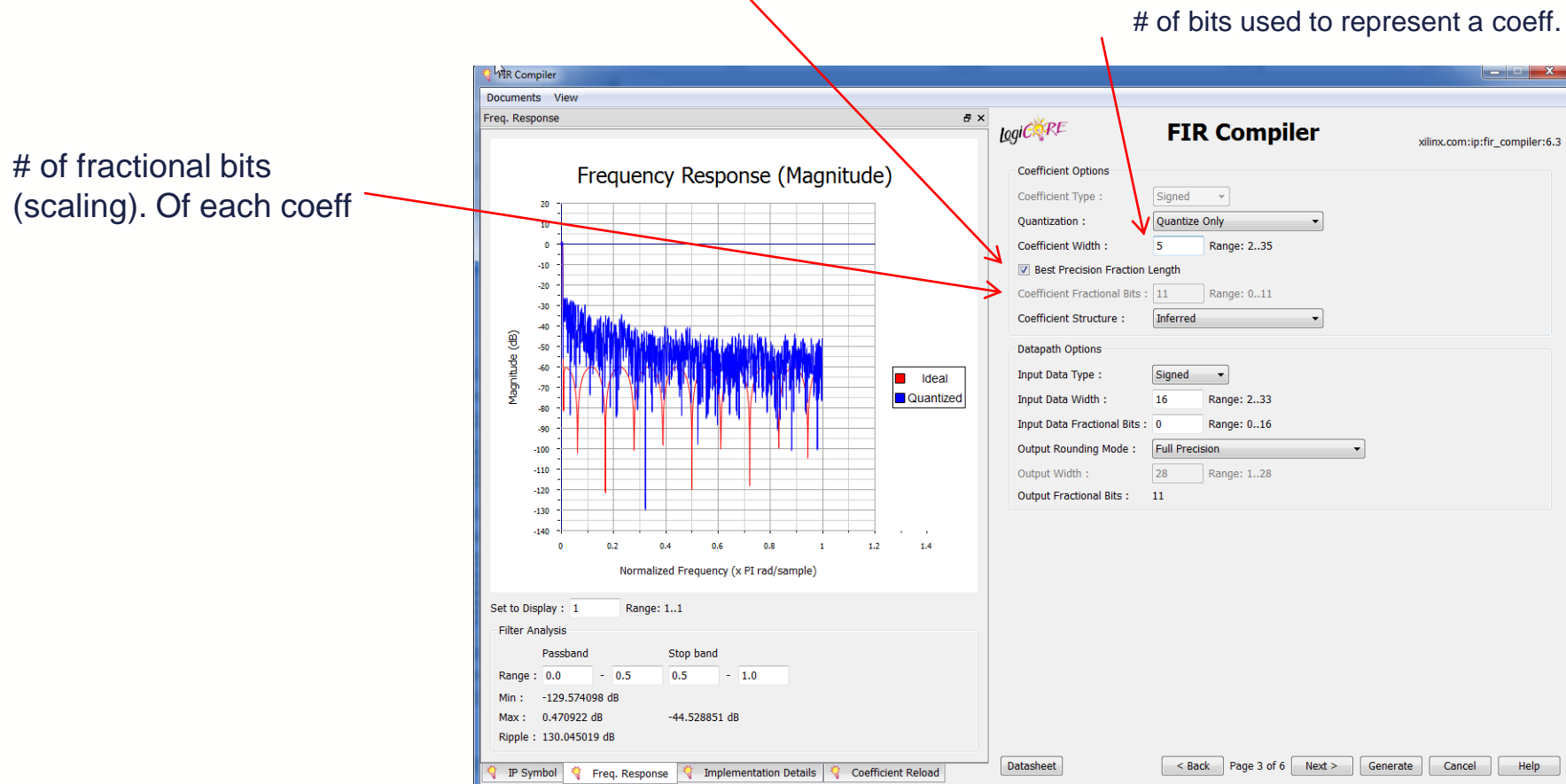
Fixed Point Number Formats

- “Q” notation is used as a standard means of communicating the implicit scaling of a number.
 - QX.Y : This X+Y bit number should be interpreted as having X integer bits and Y fractional bits.
In other words, this number is actually your true number multiplied by 2^Y
Ex. Q4.4 place values : 8 | 4 | 2 | 1 | .5 | .25 | .125 | .0625
 - Sometimes the X is left out, and the format is just named QY, communicating the number of fractional bits, but not the overall size
- Ex : x”1001” in Q16.0 = 4097. Q14.2 = 1024.2
- Provides a convenient means to track scaling throughout the signal processing chain



Scaling and Quantizing $h[n]$

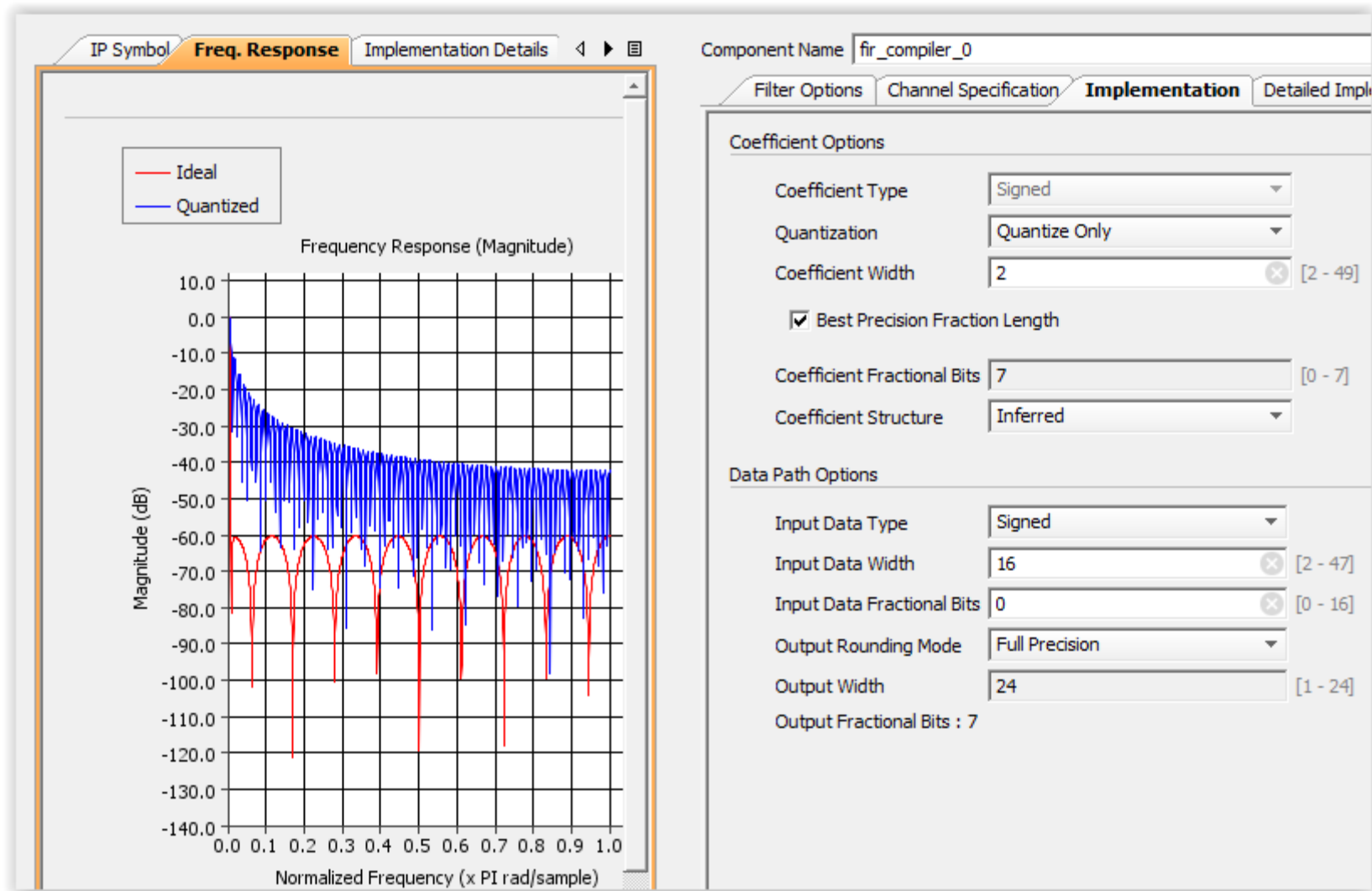
FIR Compiler provides a means for us to input floating point coefficients, and graphically choose (or let it choose) the scaling

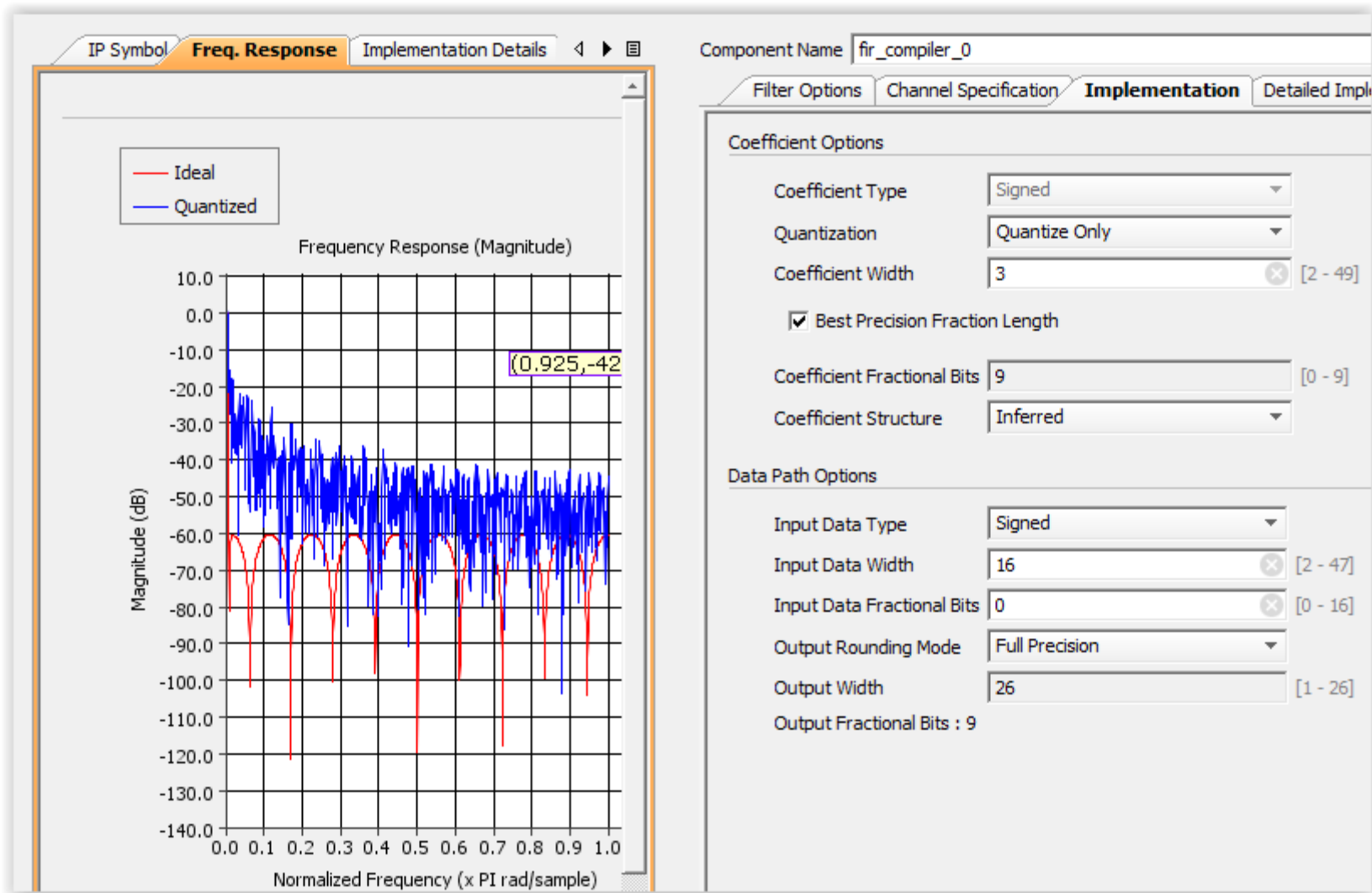


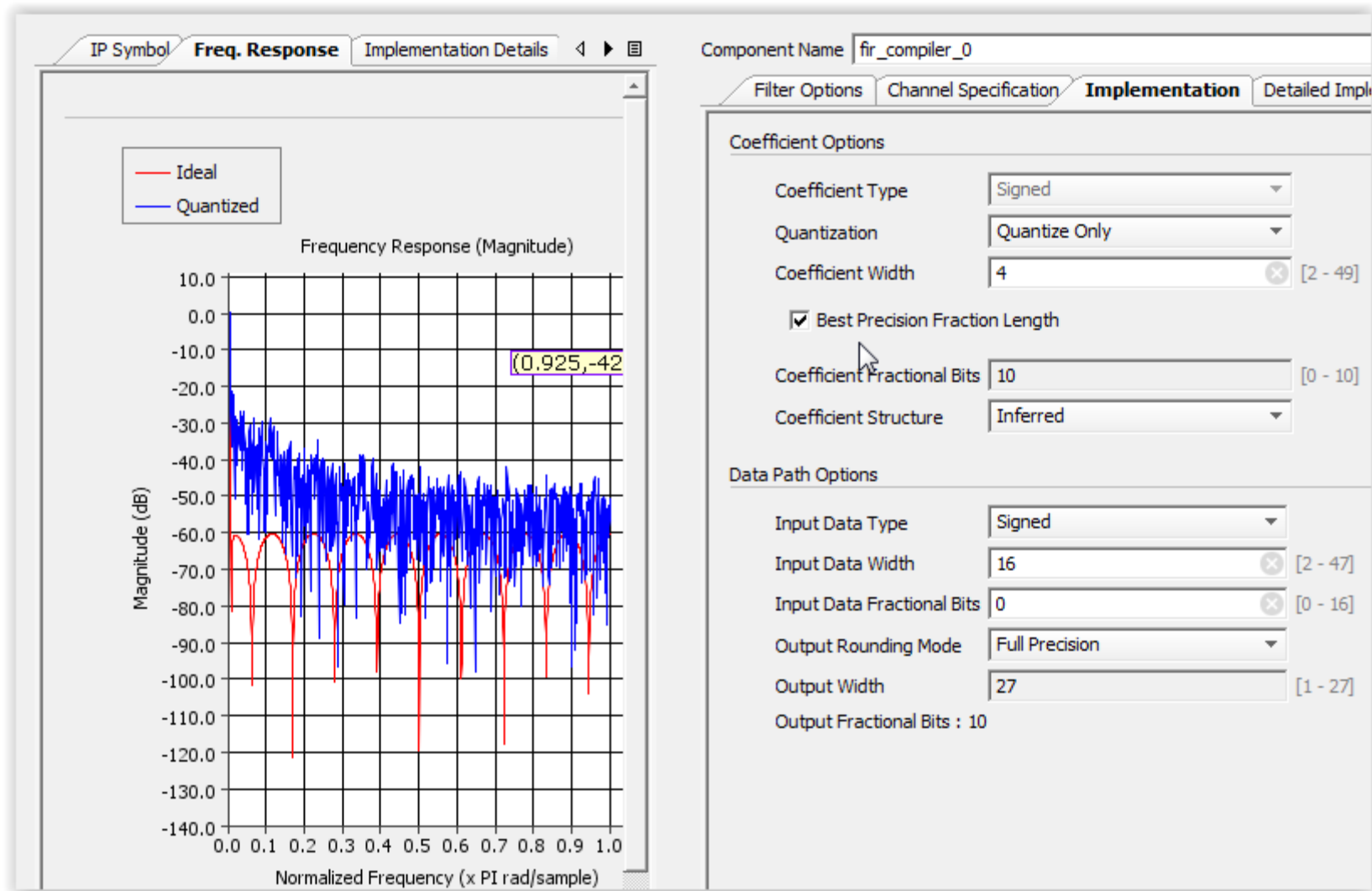
Scaling and Quantizing $h[n]$

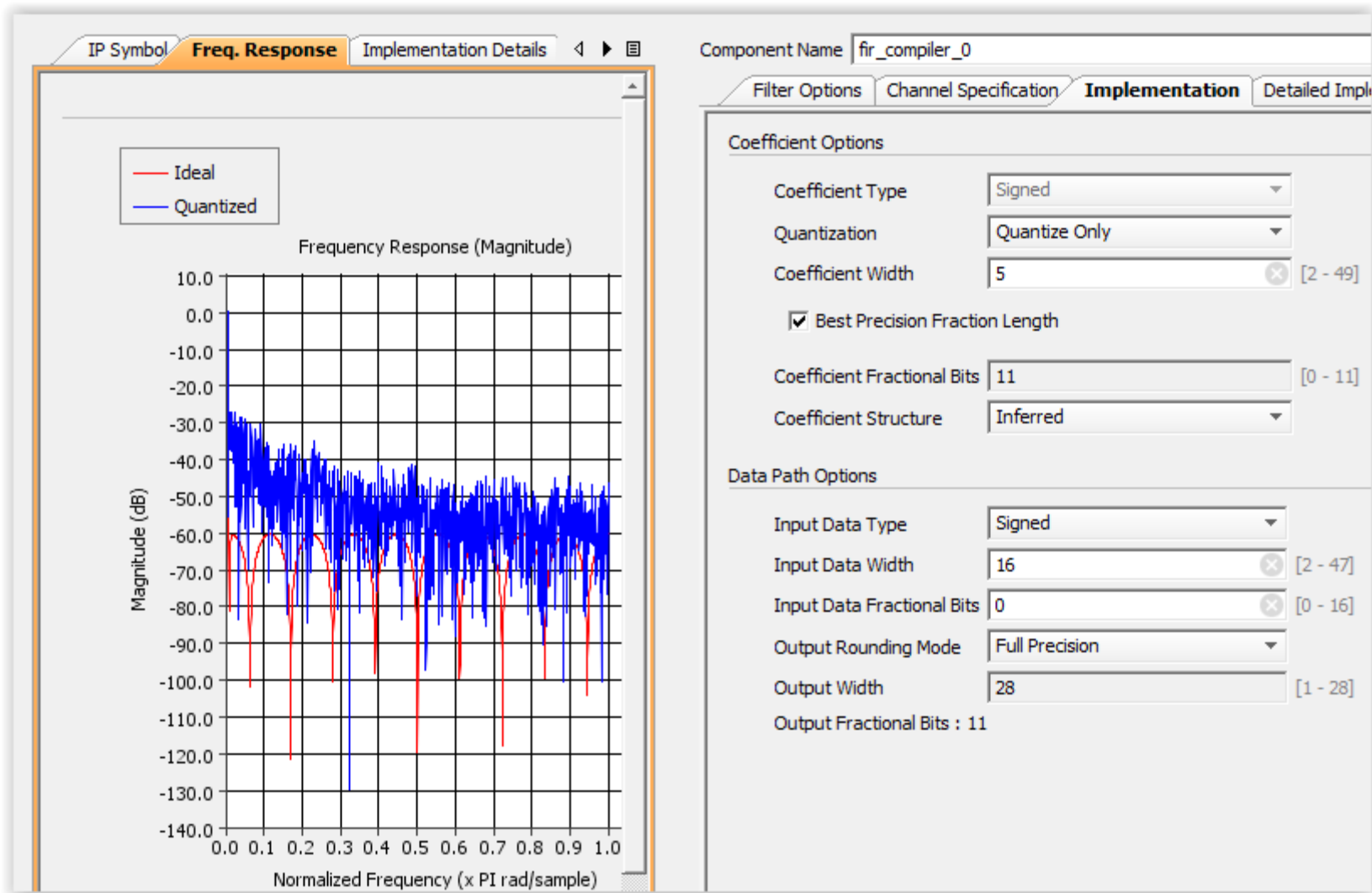
- In this method, scaling is done by IP customizer, which tells you how much it scaled the coefficients by
 - # of fractional bits is shown on the bottom, which tells you how to interpret result. Filter itself has same gain as Matlab (unity) _if_ you interpret the numbers that way
 - Effects of coefficient quantization can be observed easily during design

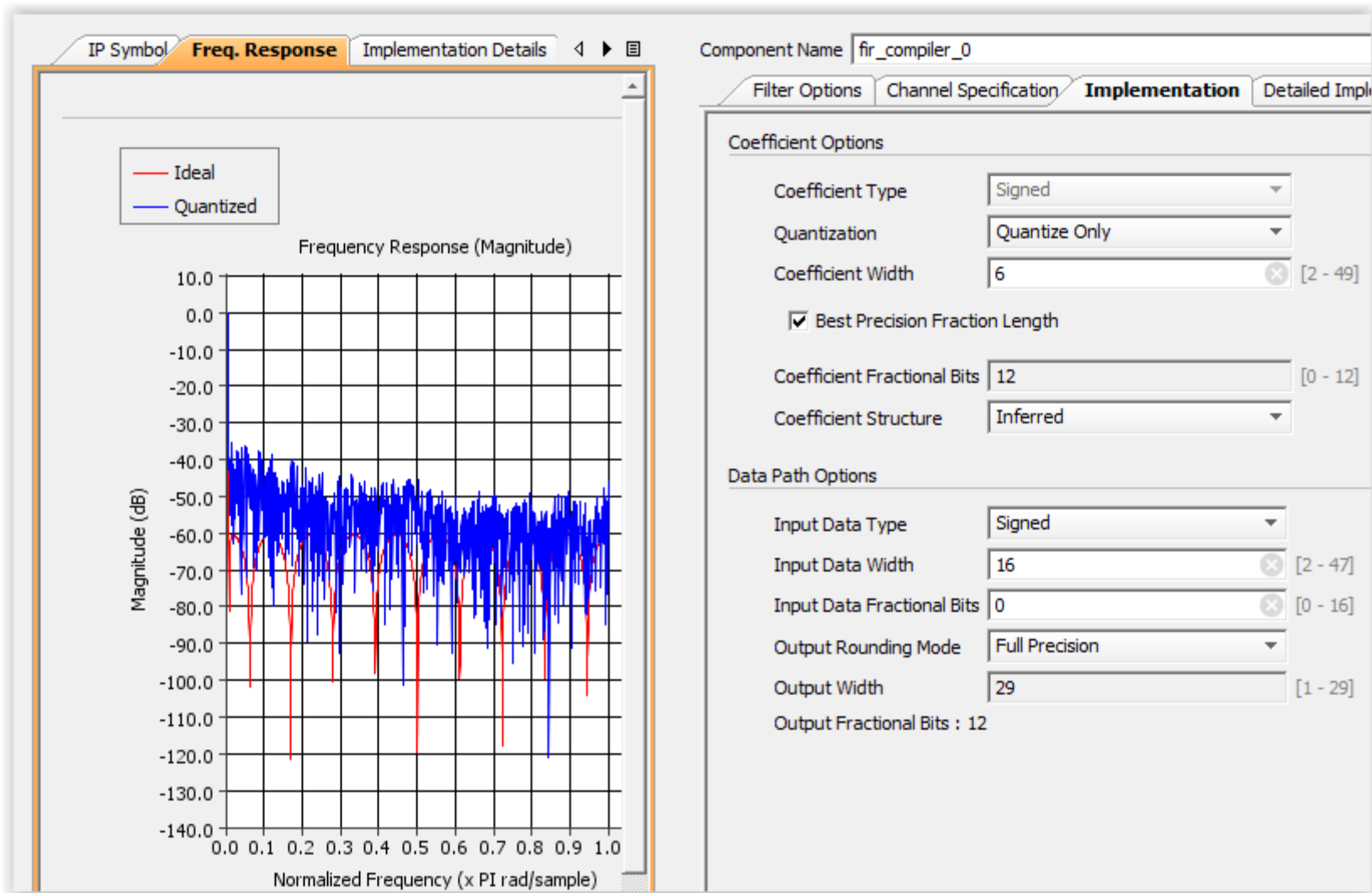


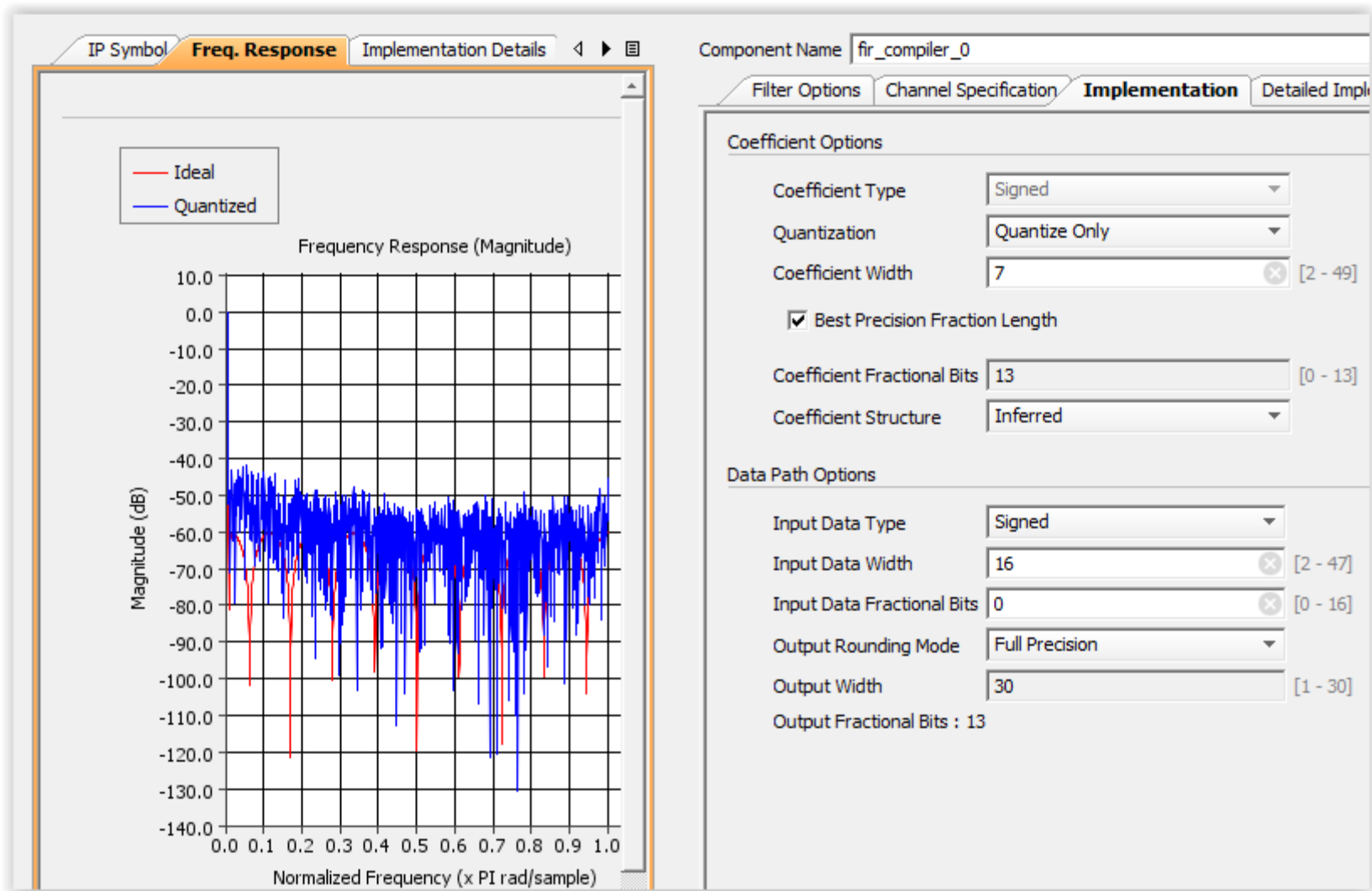


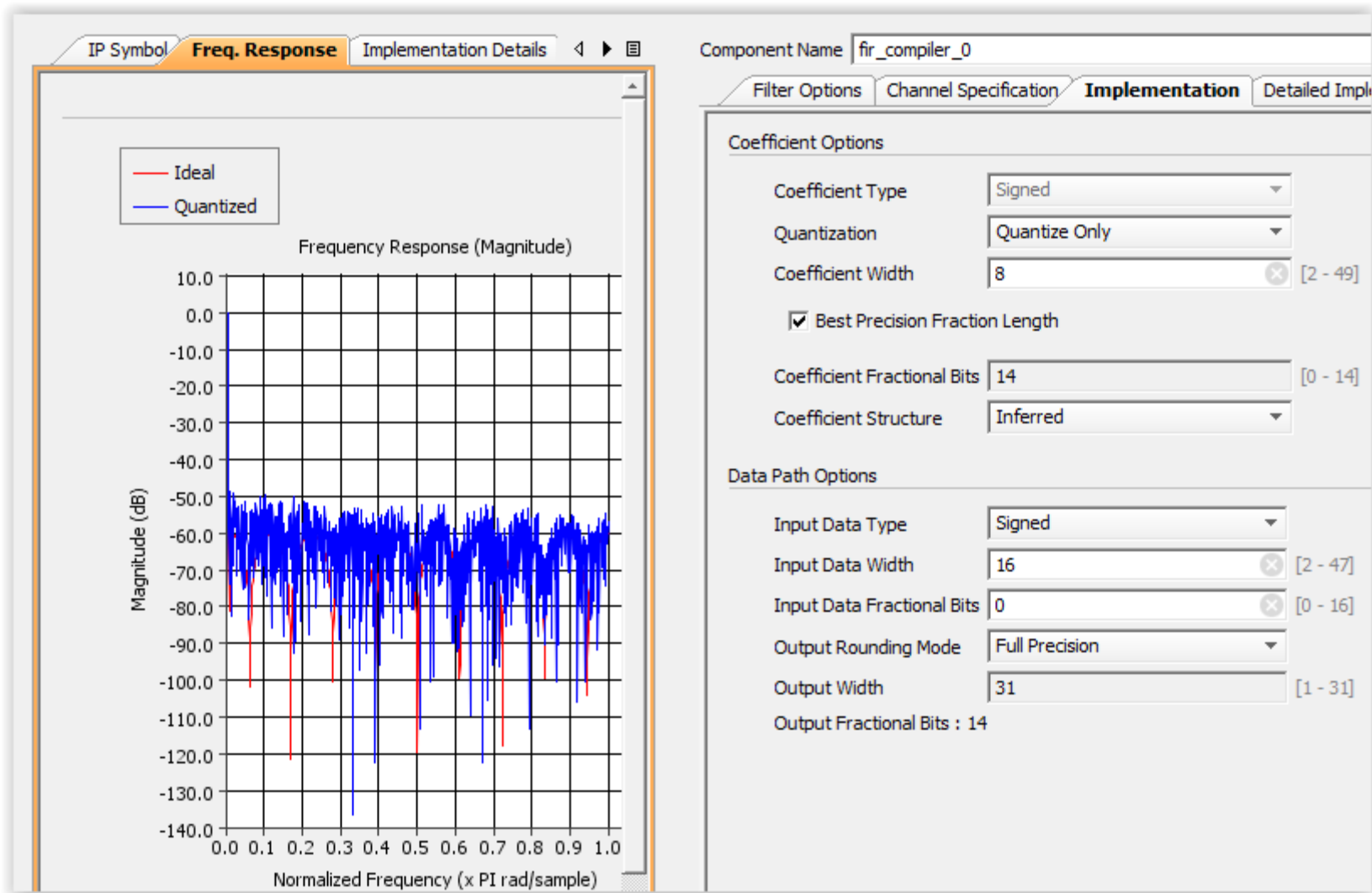


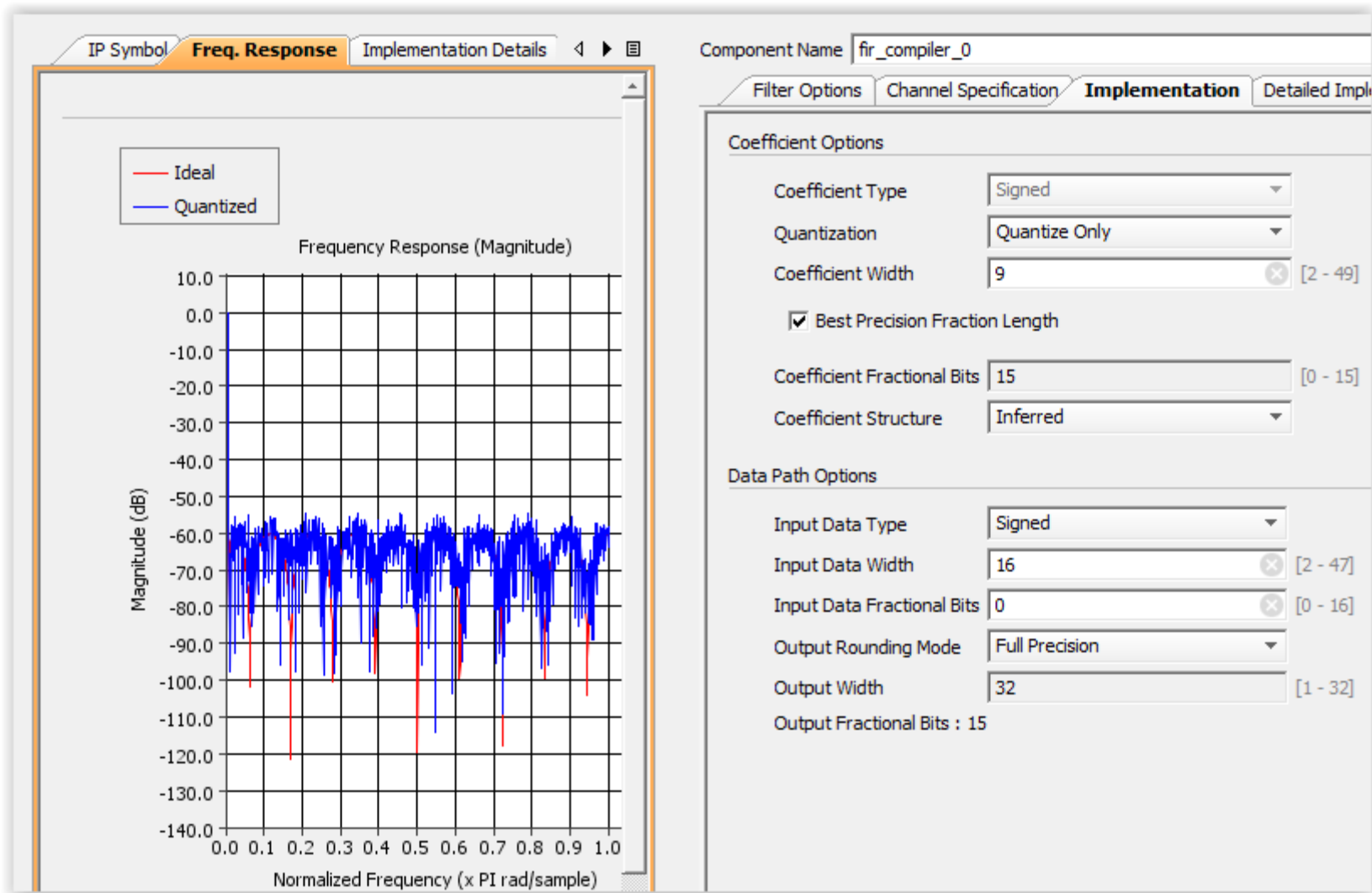


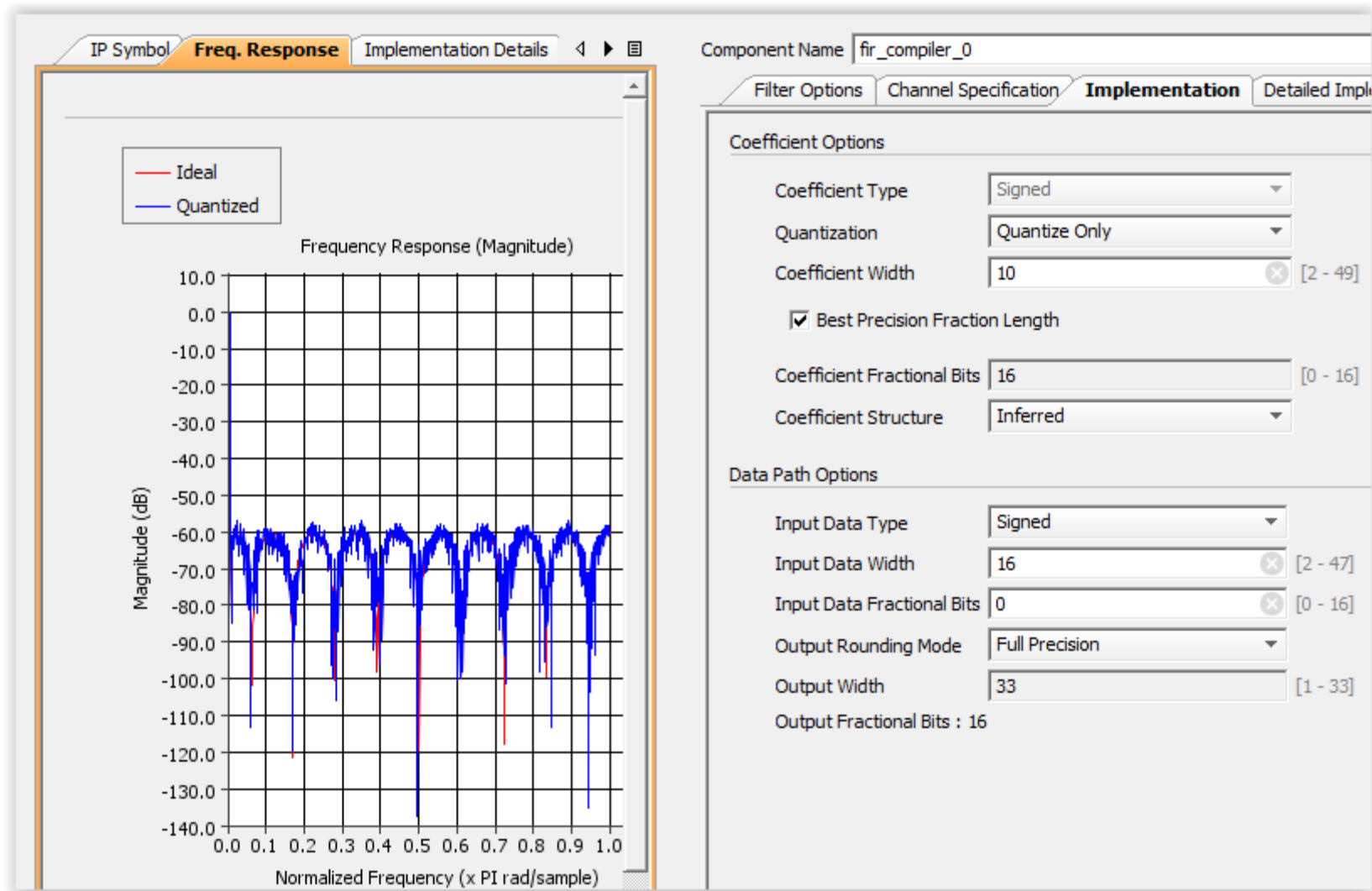


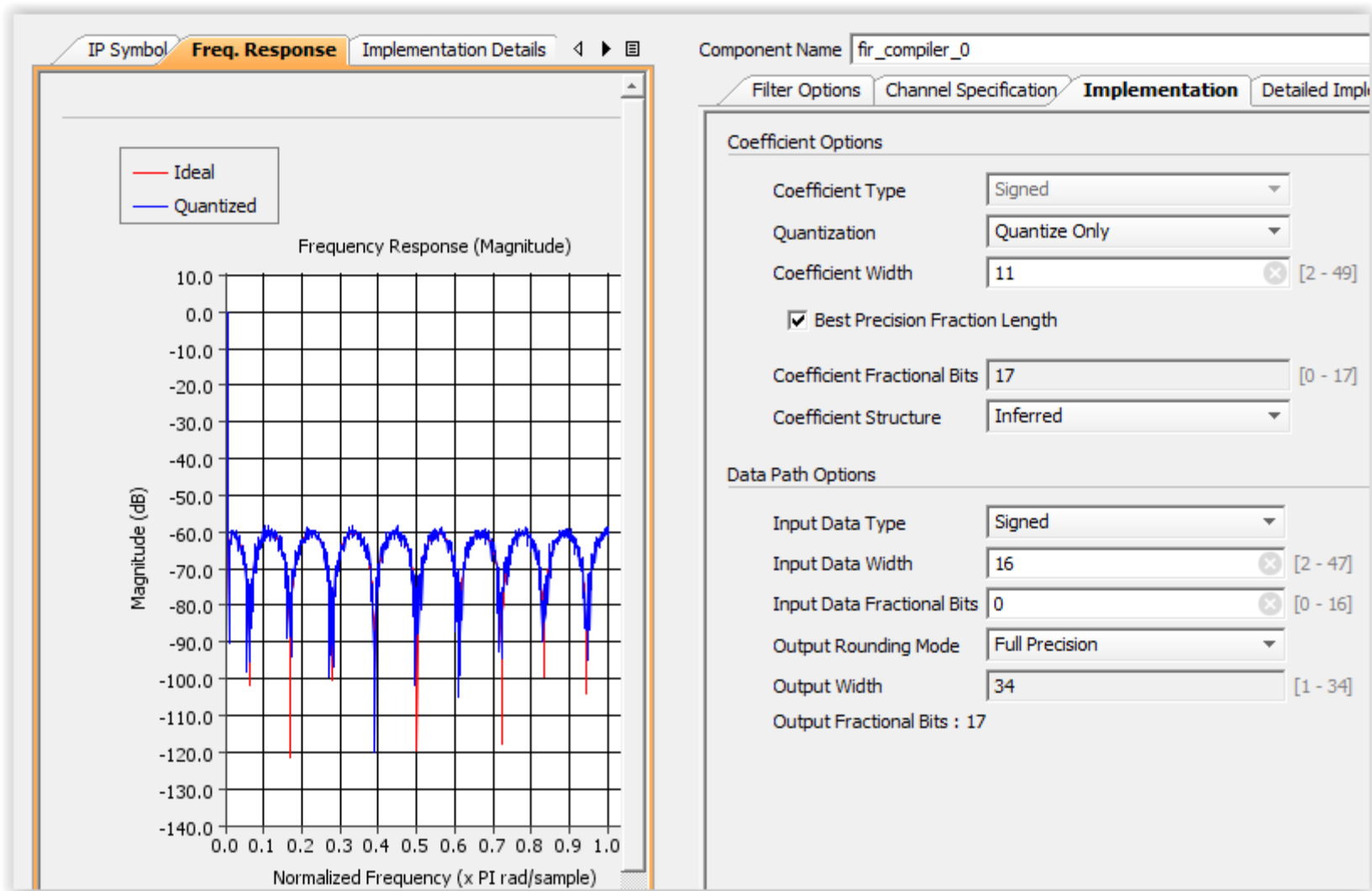


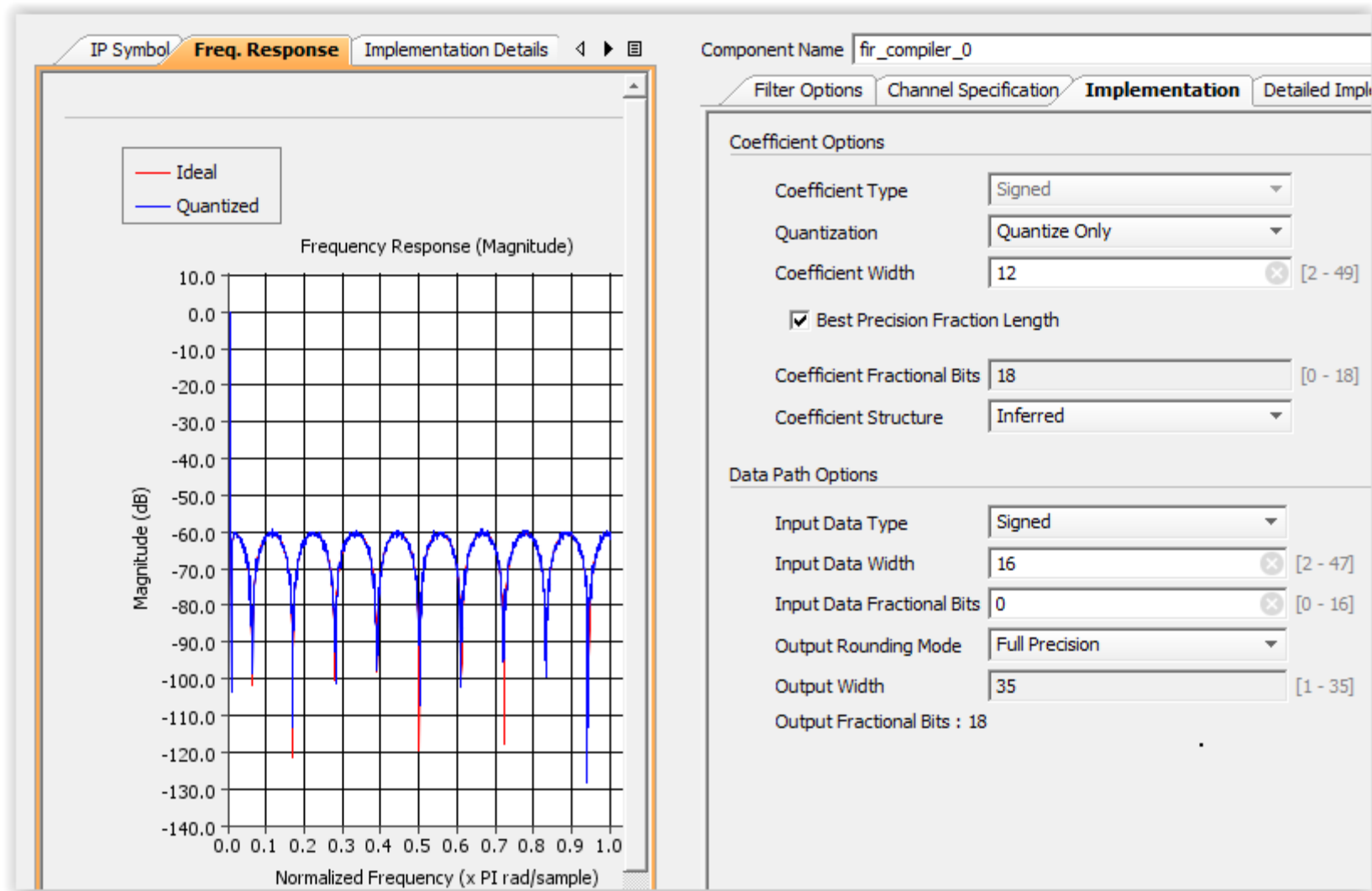


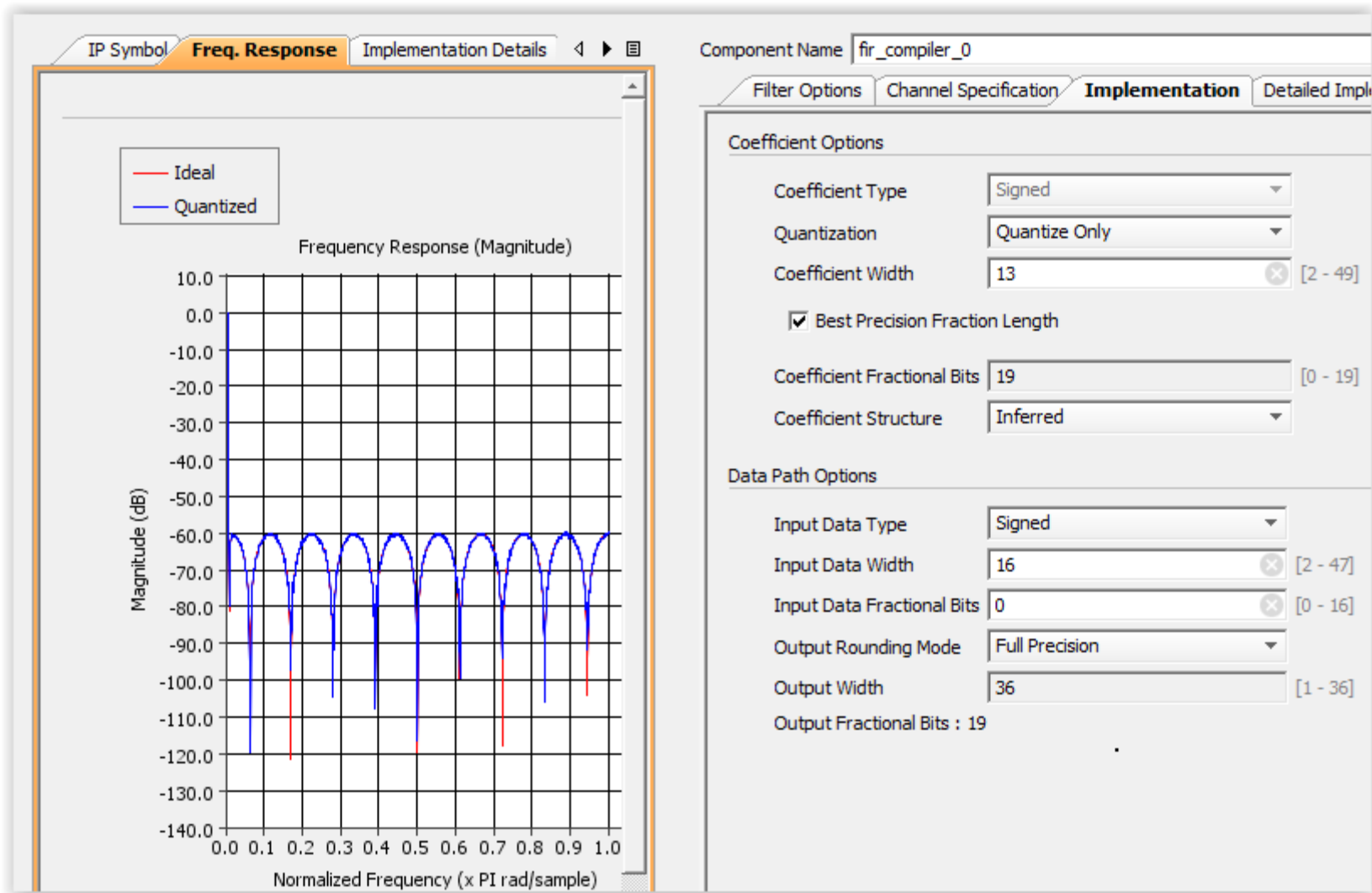












Interpreting Output

- In full precision mode using scaling factor or “number of fractional bits” to make decision on wisest bits to preserve
 - Eventually we want 16-bit outputs, that is what the DAC takes
- Filter core can also “round” for you, you specify output width and it will oblige
 - Make sure you understand result numerically

Output will be scaled such that maximum value out of the filter can never overflow

of fractional bits in output is shown. NOTE : it will never go lower than 0.



Decimating FIR Filter Core (“AXI-Stream Interface”)

When filter is busy computing this will be low. If you’ve kept your previous “promise”, this will never be low when you are giving it a new sample



What is relationship?
(latency, frequency?)

Lab Details

- Lab is broken into two parts to emphasize a good design practice
- Lab 4-A
 - Design of the filter itself
 - Using the Xilinx provided bit accurate model for the FIR compiler to choose parameters for the core that produce the desired results
 - Characterizing the performance of the filter with various inputs so that we can know exactly what to expect when placing this in the FPGA
 - Due next week
- Lab 4-B
 - Placing the core into your DDS Lab design (no software changes required)
 - Confirm with a quick ILA experiment that the results match what your model says will happen
 - Due two weeks from today

