

Finite Impulse Response Digital Filters

Practical Approaches

Specifications to fully implemented design in hardware.

First Edition

February 2003

©Signal Processing Group Inc.

*By the technical staff of:
Signal Processing Group Inc.
561 E. Elliot Road, #171
Chandler
Arizona 85225
[Email: spg@signalpro.biz](mailto:spg@signalpro.biz)
[Web: www. Signalpro.biz](http://www.Signalpro.biz)*

Introduction

This e – book is being written as a result of the experience we have had of trying to put together a system for the design of Finite Impulse Response digital filters or FIR filters for short.

We know there are countless books, articles and web pages on these filters for an interested person to look at. All these resources are extremely useful and have been of tremendous help to the author in putting this e – book together.

The author acknowledges the contribution of these wonderful people who have expended so many hours of labor to put information on the web and in books at very low cost. We thank them.

So what is the difference between this and the other books?

What we have tried to do here is produce a treatise for the practicing engineer which allows him to set up a practical system for designing FIR filters from scratch (read concept) to a fully functioning FIR filter using custom hardware.

This is different from producing code for a standard Digital Signal Processor (DSP) or a computer program that does filtering in non – real time. Or, indeed, using packaged routines for FIR filter implementation.

This is a book which allows a practitioner to design and implement a FIR filter using hardware gates, amplifiers and primitive components. Granted that this will not be something that everyone will want to do. However, if the user cannot use a standard DSP or off line code then this e – book will come in really handy!!

We took a concept and followed through with the paper design, the iterations, and finally the hardware for a set of filters. The implementations are shown in as much detail as possible. Wherever possible the software used was freeware or shareware available on the web, with acknowledgements. Some of the software is not freeware, specially for simulation of logic or circuits.

However, anyone can obtain this software in demo form for checking it out before purchase from industry standard vendors. Also the user does not have to use the software we used.

Any software which does the job is fine. What is important is to get the principle across clearly. Finally test results of the actual hardware implementation (PCB, FPGA etc) are also given and conclusions drawn.

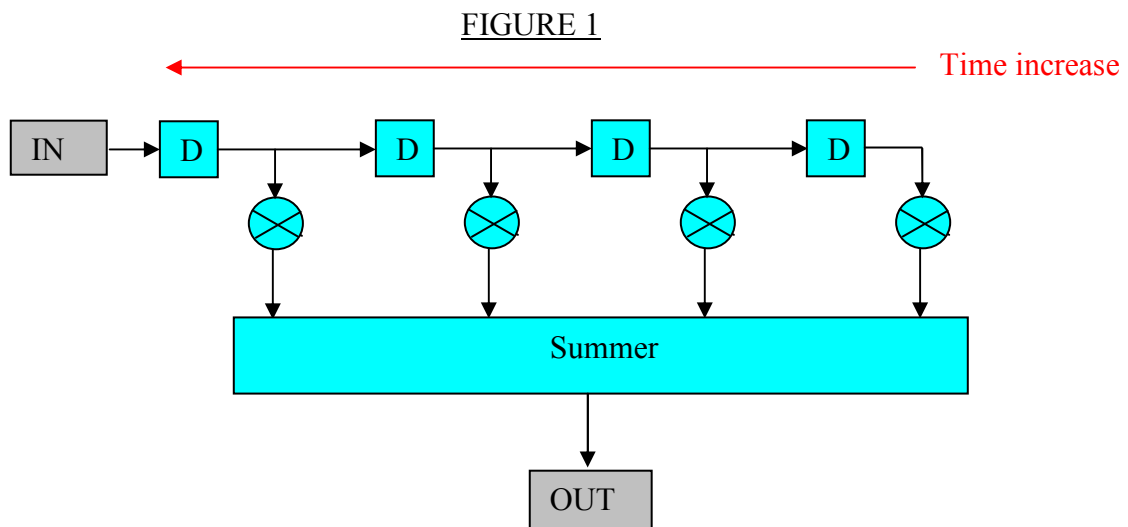
Good luck with your design!! Any questions can be directed to the author by email.

Definition of the FIR Filter

The FIR filter is a circuit that filters a digital signal (samples of numbers) and provides an output that is another digital signal with characteristics that are dependent on the response of the filter.

This is what all digital filters do. However, the FIR filter has the following differentiating characteristics:

- The FIR filter is non – recursive: It uses a finite duration of non zero input values and produces a finite duration of output values which are non zero.
- FIR filters use addition to calculate their outputs just like averaging does.
- The primitive elements used in the design of a FIR filter are delays, multipliers and adders.
- The FIR filter consists of a series of delays, multiplications and additions as shown below in the FIGURE 1 to produce the time domain output response.
- The impulse response of the FIR filter is the multiplication coefficients used.
- The phase of a FIR filter is linear. This is a dominant feature of the FIR filter.
- The frequency response of the FIR filter is the DFT (Discrete Fourier Transform) of the filter's impulse response.



A 4 tap FIR filter

Understanding the Operation

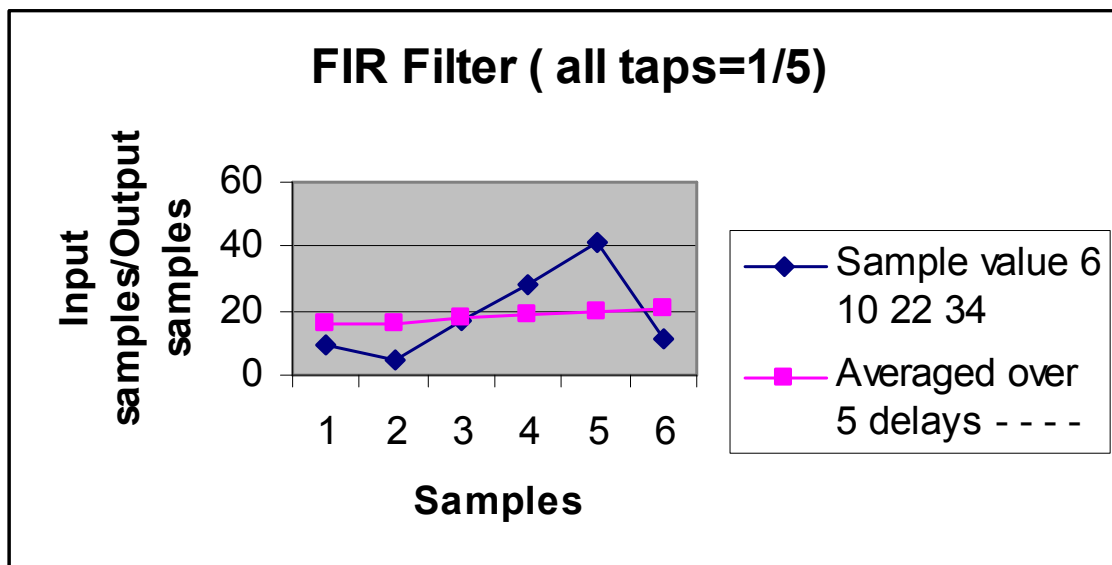
To understand the operation consider TABLE 1 which shows the input samples, the time delays for those samples, and the FIR operational result over 5 delays with a constant multiplier of 1/5 (Note this is simply an averaging operation!).

TABLE 1: Averaging operation as a FIR filter

Delay (Time index)	Sample value	Averaged over 5 delays
1	6	-
2	10	-
3	22	-
4	34	-
5	9	16.2
6	5	16.0
7	17	17.4
8	28	18.6
9	41	20.0
10	11	20.4

In the Plot below the sample values and the average sample values over 5 time delays are shown with the time delay as the independent variable.

PLOT 1: Low pass FIR operation with fixed tap weights of 1/5



Note that the filter has smoothed out the sharp variations in the values of the input samples. This is nothing less than a low pass FIR filter.

The reason is that fast changes imply high frequencies in the frequency domain. Since the filter has rejected the fast changes in sample values it has attenuated the high frequencies leaving only the low frequency components at the output.

In this case all the taps or multipliers were set to a values of 1/5. In other words what the filter did was as follows:

nth Output value = $y(n) = 1/5[x(n-4) + x(n-3) + x(n-2) + x(n-1) + x(n)]$ where $x(k)$ is the input sample value. (1)

In terms of FIR format this should really be written as:

$y(n) = 1/5[x(n-4)] + 1/5[x(n-3)] + 1/5[x(n-2)] + 1/5[x(n-1)] + 1/5[x(n)]$ (2)

In other words the FIR operation is delay, multiply, add!

It should be obvious from this description of the FIR operation that as the tap weights change or the number of delays change (i.e as the number of taps and their values change) the response of the FIR system also changes.

In fact this is what is done in FIR filters.

Designing FIR Filters

With this basic understanding we can actually move on to the techniques and tools of FIR filter design. The challenge is: Given a filter response characteristic design a FIR filter to meet those characteristics.

Lets start with a low pass FIR filter design.

As is usual, filter characteristics are specified in the frequency domain. The task of the FIR filter designer is to chose the right configuration and the number of taps and their values so that the designed filter meets the specifications.

There are two main techniques that are used to design FIR filters. These are the window method and the optimum method. Both of these are explained below and used in the designs to follow.

Window method:

The window method of FIR filter design is explained below by considering a continuous low pass filter. Let the frequency response of this filter $H(f)$ be ideal. This simply means that at low frequencies the response is unity. At high frequencies (beyond some frequency) the output from the filter is zero. This frequency is called the cutoff frequency.

The discrete form of this response can be obtained in a straightforward manner. The discrete response is obtained by sampling the continuous response with a sampling frequency f_s . This response is shown in FIGURE 2.0 below:

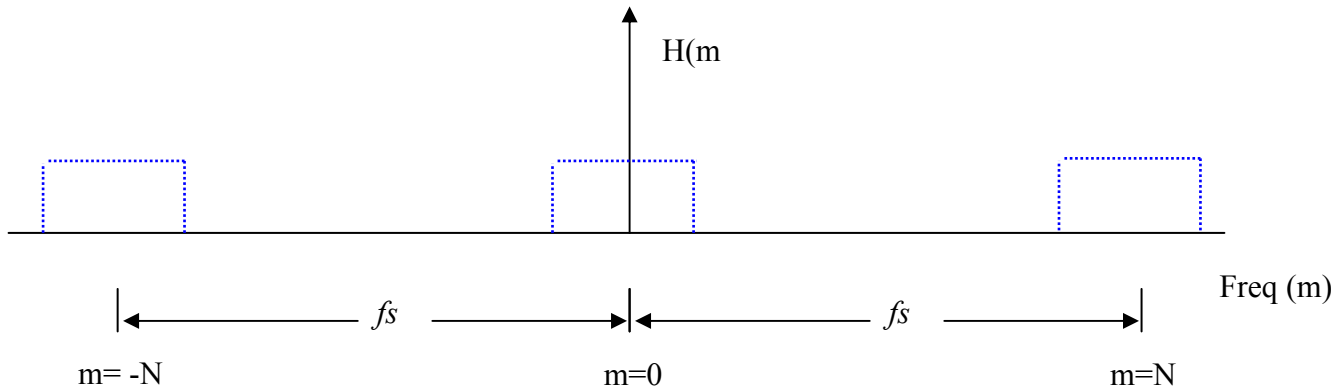


FIGURE 2.0 The discrete frequency response

The discrete frequency response is always periodic, with the period being equal to the sampling rate. This is the general case for discrete frequency domain representations. [See Appendix 1 for details.]

Now, if we are to design a FIR filter to meet these specifications we must find the number of taps and the values of the multipliers. In other words, the impulse response of the filter.

There are two methods to get the impulse response. The first one is the complicated one and rarely used practically. This is:

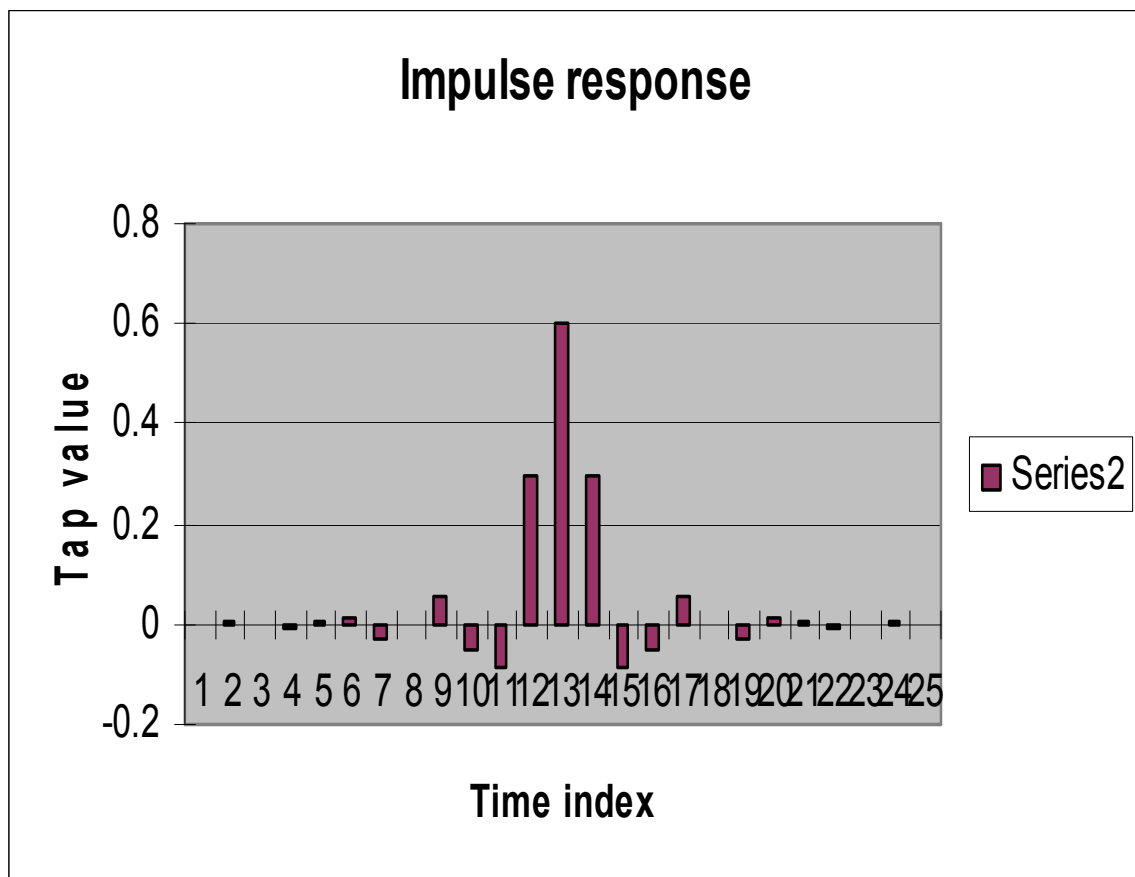
- (a) Express the discrete frequency response algebraically
- (b) Do an inverse DFT to get the impulse response
- (c) Evaluate the impulse response over a time index.

This is a difficult thing to do. A simpler method is to use freely available software to do the DFT and generate the impulse response $h(k)$.

Defining the discrete frequency response over 0 to f_s is the best way, if the inverse DFT is to be used.

The plot below, PLOT 2.0 is an example of the tap weights (impulse response) of a low pass filter with pass band at 2400 Hz and a sampling frequency of 8kHz. This response was calculated using a simple software program. The number of taps were 25.

Plot 2.0 Tap weights for the example filter



The tabular listing of the impulse response is shown below in Table 2.0.

Note that the impulse response is symmetrical around the peak value. This ensures that the phase of the filter will be linear.

The form of the impulse response is a Sinc function ($\text{Sinc}(x)$).

Table 2.0 Impulse response for the example low pass FIR filter

No	Weight
1	-0.00125
2	0.002633
3	0
4	-0.00722
5	0.007251
6	0.011253
7	-0.02725
8	0
9	0.058284
10	-0.05397
11	-0.0878
12	0.298026
13	0.600081
14	0.298026
15	-0.0878
16	-0.05397
17	0.058284
18	0
19	-0.02725
20	0.011253
21	0.007251
22	-0.00722
23	0
24	0.002633
25	-0.00125

Note, The more taps we use in the impulse response the better the approximation to the desired frequency response. This is demonstrated below: First the magnitude and phase response of the original filter is shown followed by responses with fewer and fewer tap weights.

Plot 4. Original Filter response

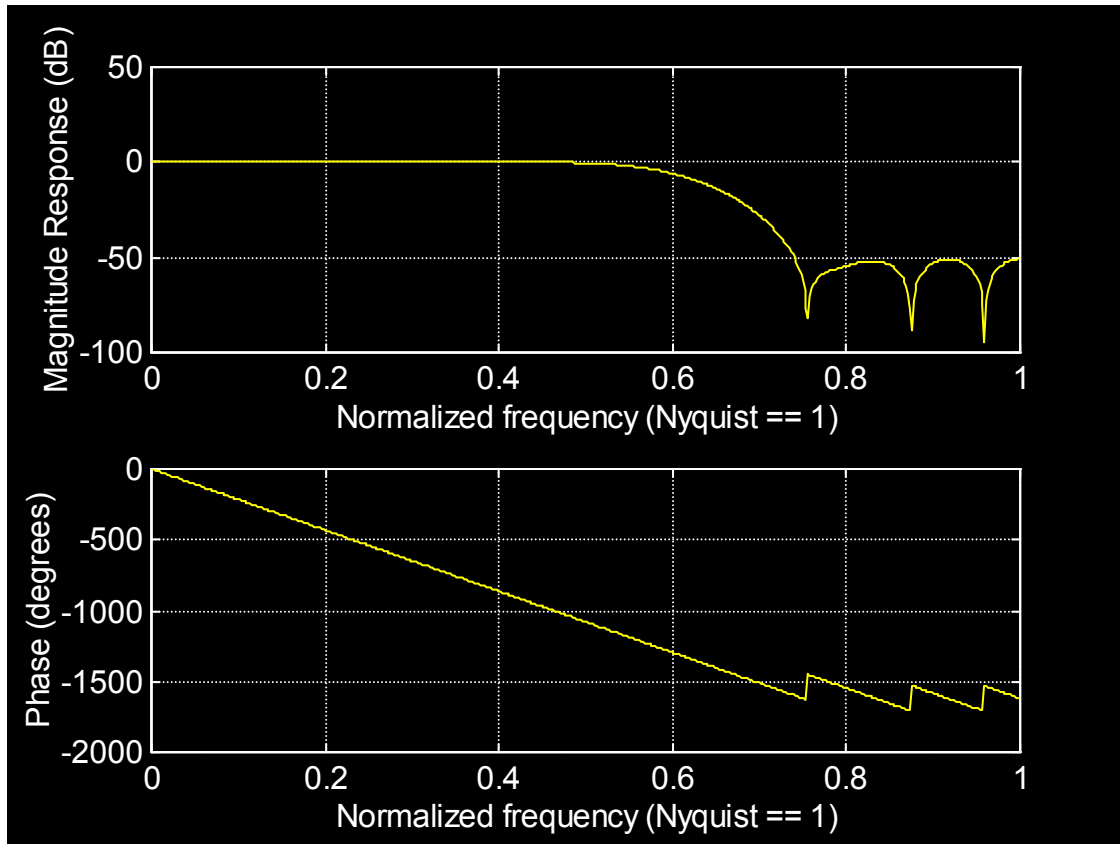


Table 4 Reduced number of tap weights

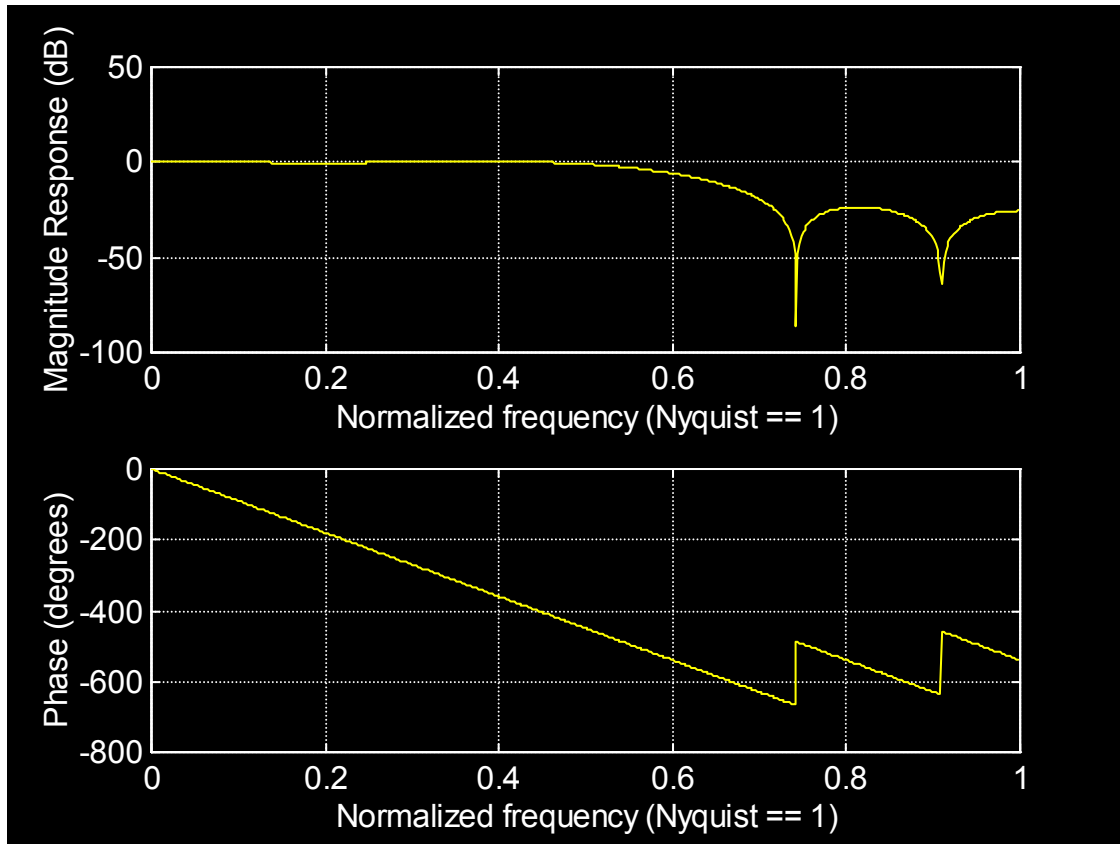
```

0
0.058284
-0.05397
-0.0878
0.298026
0.600081
0.298026
-0.0878
-0.05397
0.058284
0

```

The new response is shown below in plot 5. Notice the deterioration of the magnitude response and the effects on the phase.

Plot 5. Reduced number of taps



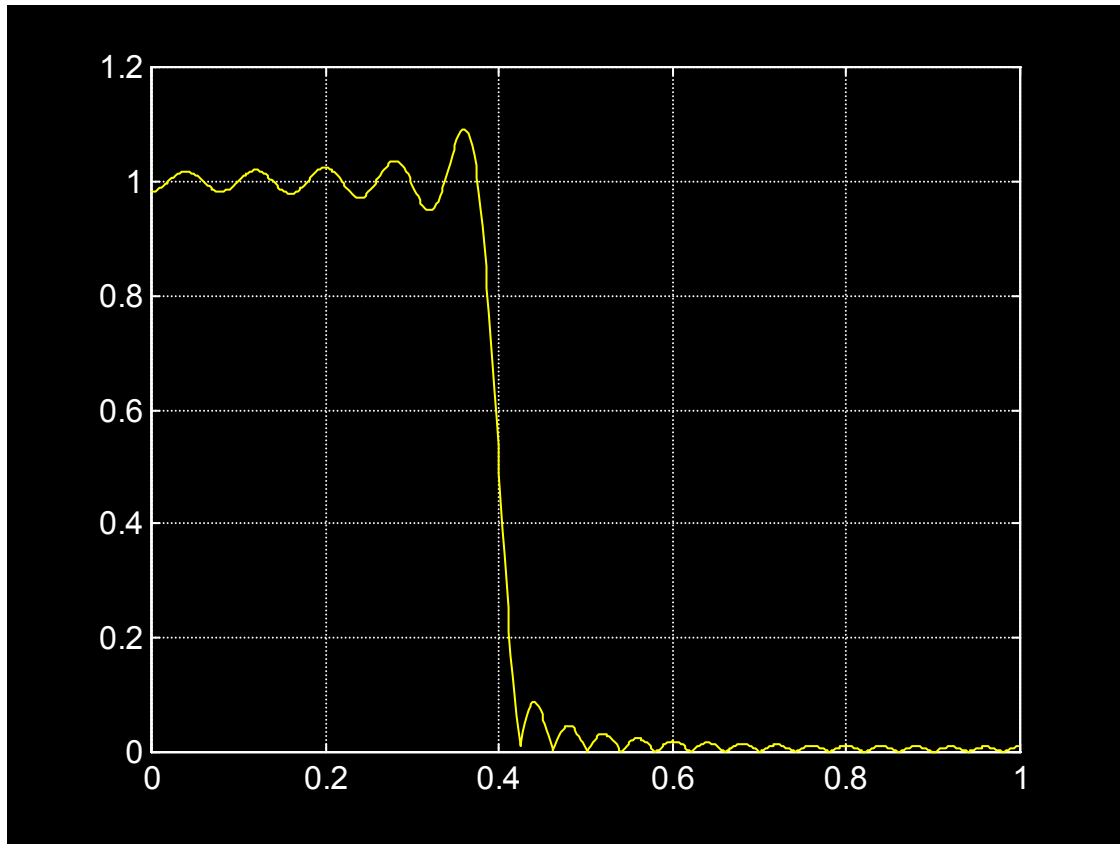
This reduction in the number of taps is a rudimentary windowing operation. We truncate the number of coefficients using another function, $w(n)$. Depending on the nature of this function we get various effects on the magnitude and phase of the filter. So using these windows functions we generate a new impulse response:

$$h^\infty(k).w(k)$$

where $h^\infty(k)$ simply represents an infinitely long impulse response (which by definition will give an extremely close approximation to the filter response we want. However, this infinitely long response is not a practical impulse response so we truncate it using the window function.

When windows are used, the effect is to create ripple in the filters response, specially near the band edges, as shown below for a rectangular window or a “boxcar” window.

Plot 6. The effect of a “boxcar” window



The filter response above was generated using a simple rectangular window of length 51 for a low pass filter with a cutoff frequency at 0.4π radians/sec (normalized). Notice the ripple in the pass band and the stop band response. This is called the Gibb’s Effect. No matter how wide the window is a windowed impulse response will always have ripples in the frequency response.

The ripple can be made larger or smaller or the sharpness of the transition between the pass band and stop band can be changed with different windows. This is where engineering judgment plays a major role.

There are a number of standard windows functions which provide differing properties. The following are the ones used most often:

- (a) Rectangular window – the simplest window specified by saying: $w(n) = 1$ when $0 \leq n \leq N-1$ where N is the total length.
- (b) Bartlett Window – In this case

$$w(n) = \frac{2n}{N-1} \quad \text{if } 0 \leq n \leq (N-1)/2 \text{ and,}$$

$$w(n) = 2 - \frac{2n}{N-1} \quad \text{if } (N-1)/2 \leq n \leq N-1$$

(c) Hanning Window – In this case

$$w(n) = \frac{1}{2}[1 - \cos(\{2\pi n\}/\{N-1\})]$$

if $0 \leq n \leq N - 1$.

(d) Hamming Window – In this case

$$w(n) = 0.54 - 0.46\cos(\{2\pi n\}/\{N-1\})$$

if $0 \leq n \leq N - 1$.

(e) Blackman Window – In this case

$$w(n) = 0.42 - 0.5\cos(\{2\pi n\}/\{N-1\}) + 0.08\cos(\{4\pi n\}/\{N-1\})$$

if $0 \leq n \leq N - 1$.

The above windows are simpler but do not give us control over the frequency response. In other words all we can do is select one or the other of these windows and accept the frequency response.

There are two other windows called the Chebyshev and the Kaiser windows which allow greater control over the response.

The Chebyshev window is defined as the inverse discrete Fourier transform of the function:

$$\text{Cos}[N.\text{Cos}^{-1}[\alpha.\text{Cos}(\pi m/N)]]/[\text{Cosh}[N.\text{Cosh}^{-1}(\alpha)]]$$

Where $\alpha = \text{Cosh}(1/N\{\text{Cosh}^{-1}(10^\gamma)\})$ and $m = 0, 1, 2, 3, \dots N-1$

This looks very complicated, however realize that, we will be using computer aided design for most of our work and these difficult calculations can be left to the software. We simply need to know what the quantities are, that we will be manipulating.

The Kaiser window is defined by the inverse DFT of the function:

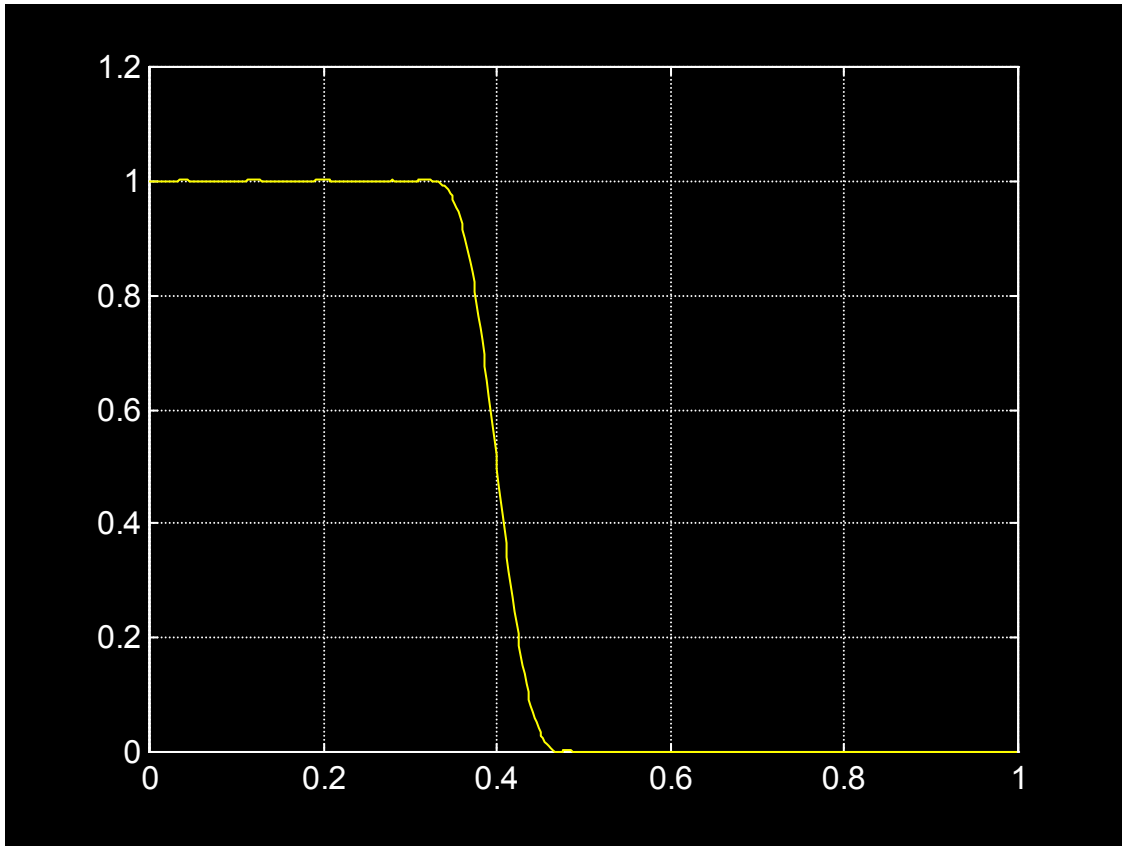
$$\text{Io}[\beta\sqrt{\{1 - [(k - p)/p]^2\}}]/\text{Io}(\beta)$$

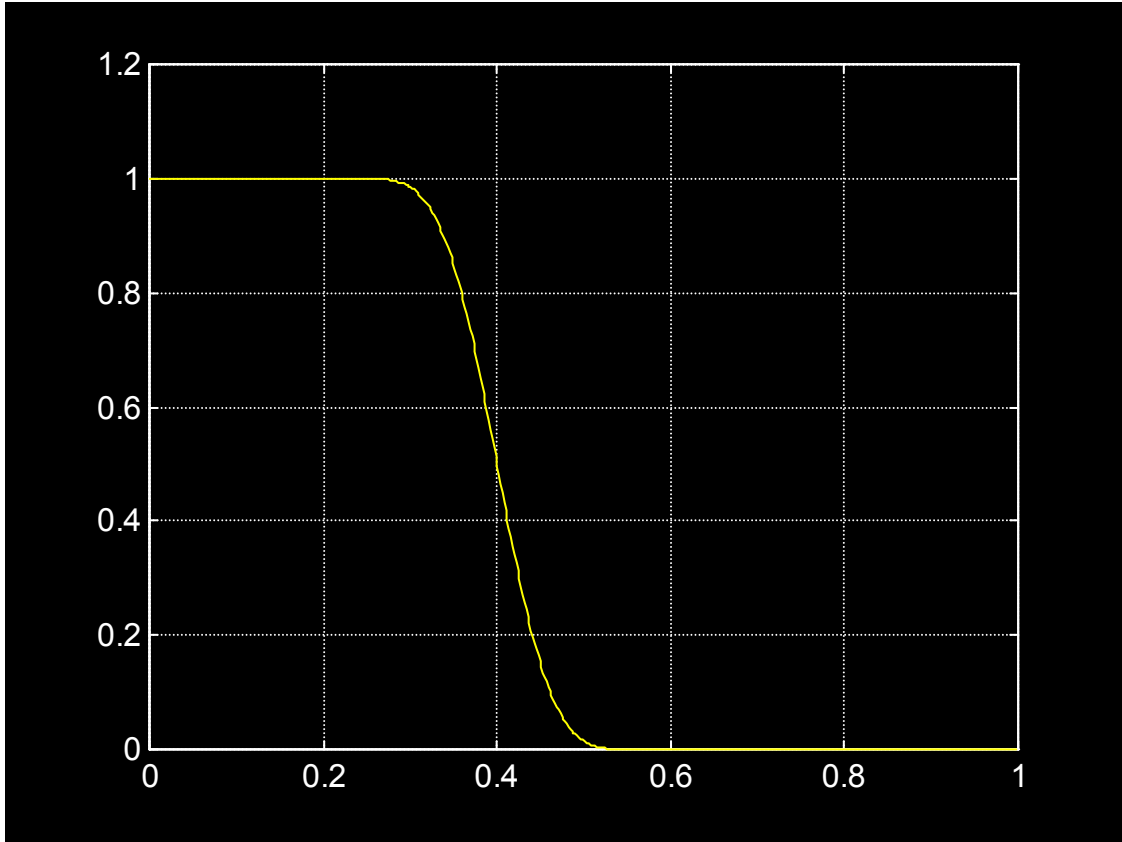
For $k = 0, 1, 2, 3, \dots N-1$ and $p = (N-1)/2$.

The fact is that we have the parameters γ and β that give us control over windows main lobe and side lobes. By using available software packages we should be able to experiment and finalize the response we want using these parameters. This will be done in the following for all the windows to see what the major effects are.

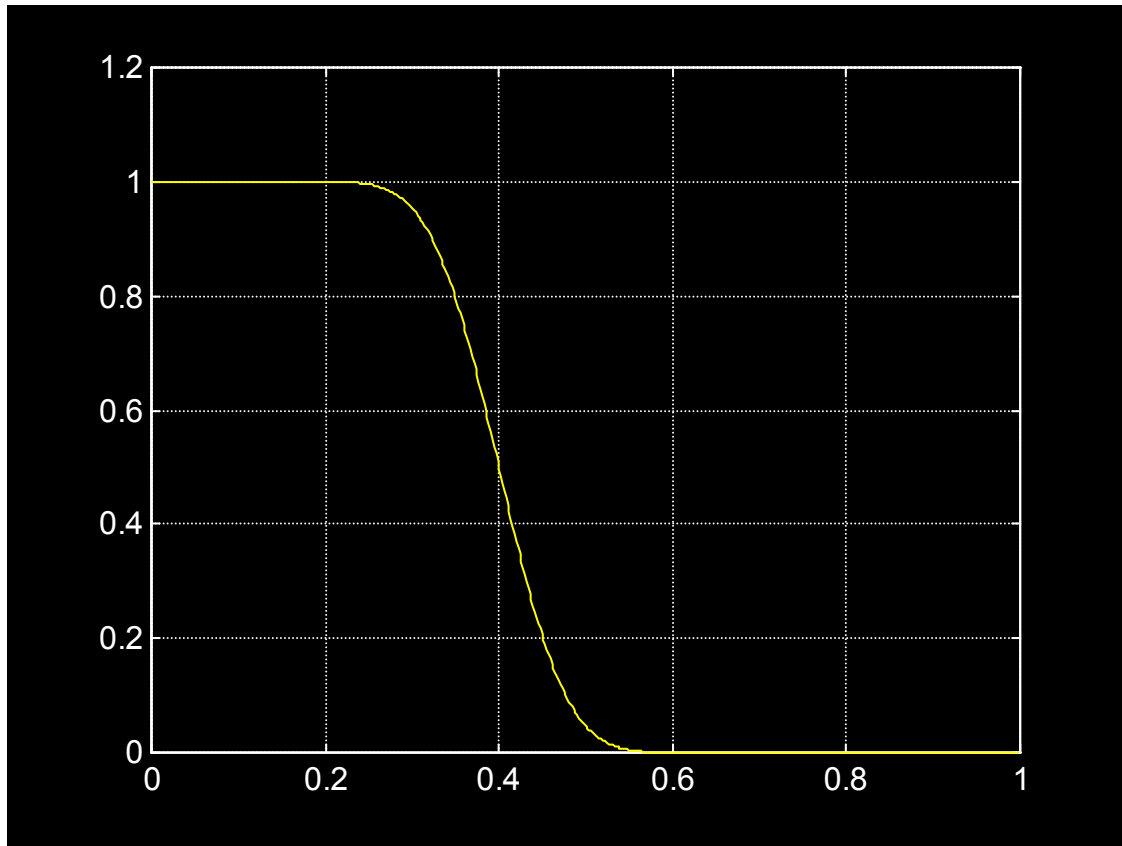
The same 51 coefficients with a few windows generates the responses shown below.

Plot 7. Hamming window used



Plot 8 Hanning Window used

Plot 9 Blackman Window



Some experiments with Kaiser windows are shown below. First of all, the parameter β is the parameter that controls side lobe and main lobe heights. A empirical rule for using β is that for a side lobe height of $-\alpha$ dB, the β parameter is:

$$\begin{aligned} \beta &= 0.1102(\alpha - 8.7) && \text{if } \alpha > 50. \\ \beta &= 0.5842(\alpha - 21)^{0.4} + 0.07886(\alpha - 21) && \text{if } 50 \geq \alpha \geq 21 \\ \beta &= 0 && \text{if } \alpha < 21 \end{aligned}$$

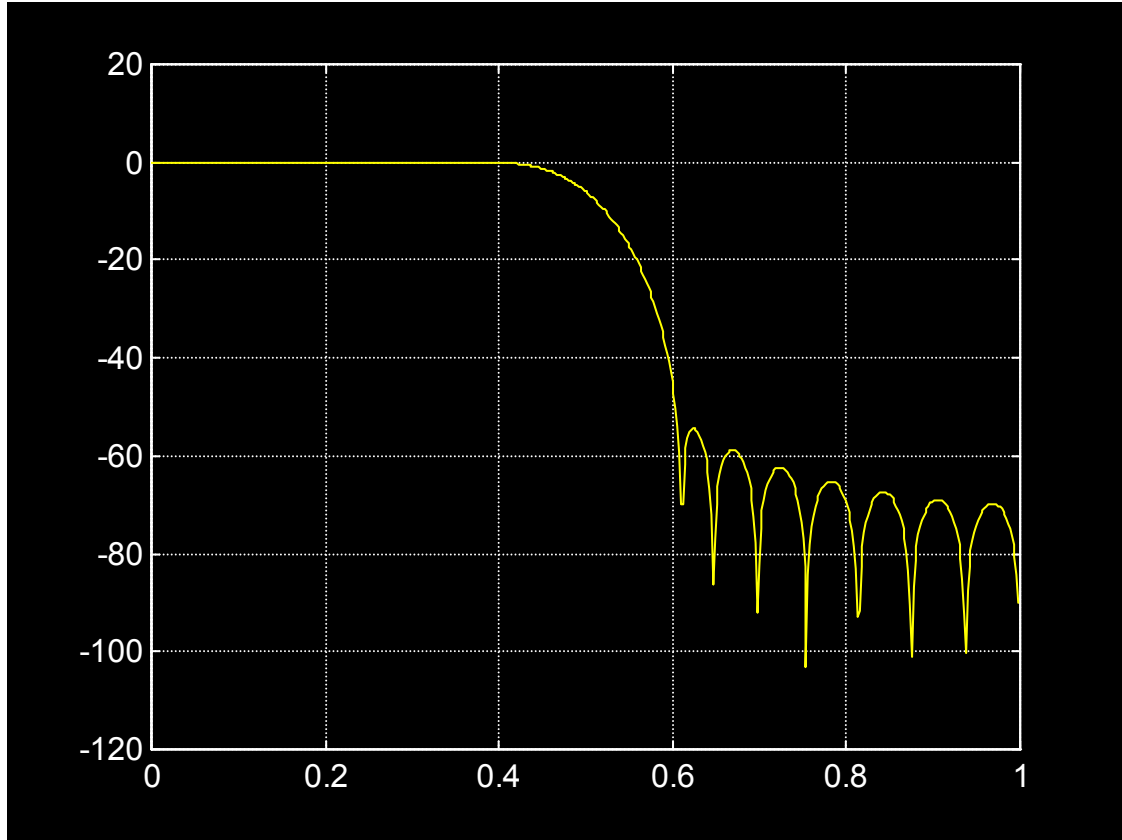
For a transition width of pass band edge to stop band edge of $\Delta\omega$ rads/sec the length should be:

$$n = [(\alpha - 8)/(2.285\Delta\omega)] + 1$$

If these approximations are used the designed filter should be very close but will still need optimizing using software (CAD) The example below shows this.

Let $\alpha = 55$ dB
Then $\beta = 5.10226$

Let transition band be $= 0.2\pi$ rads/sec, say cutoff frequency 0.5π rads/sec and 55 db attenuation, $n = 32$

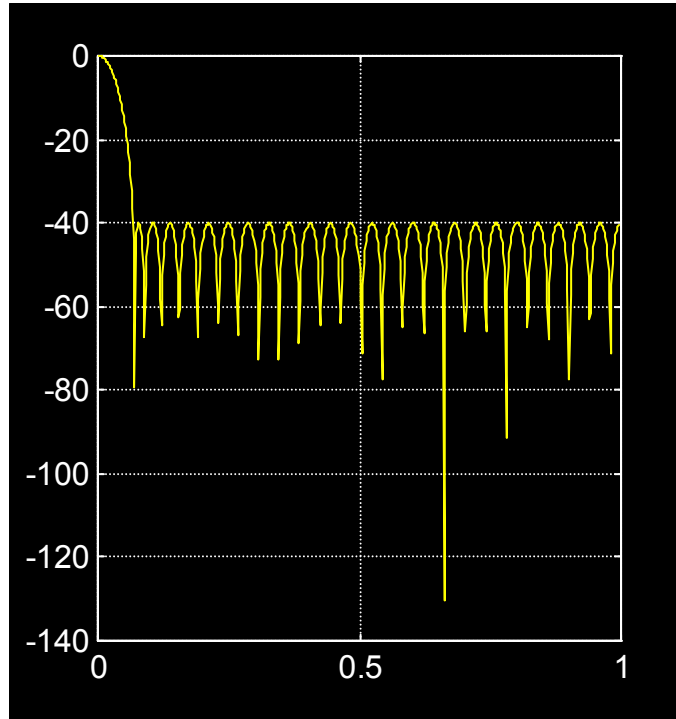
Plot 10. Low pass filter designed using Kaiser Window

It appears that the attenuation is met and the transition region is well within the requirements.

A similar experiment with Chebyshev windows can be done. The Chebyshev window minimizes the main lobe width given a particular side lobe height

Using the following parameters: $n = 51$, 40 dB side lobe height we get the following response:

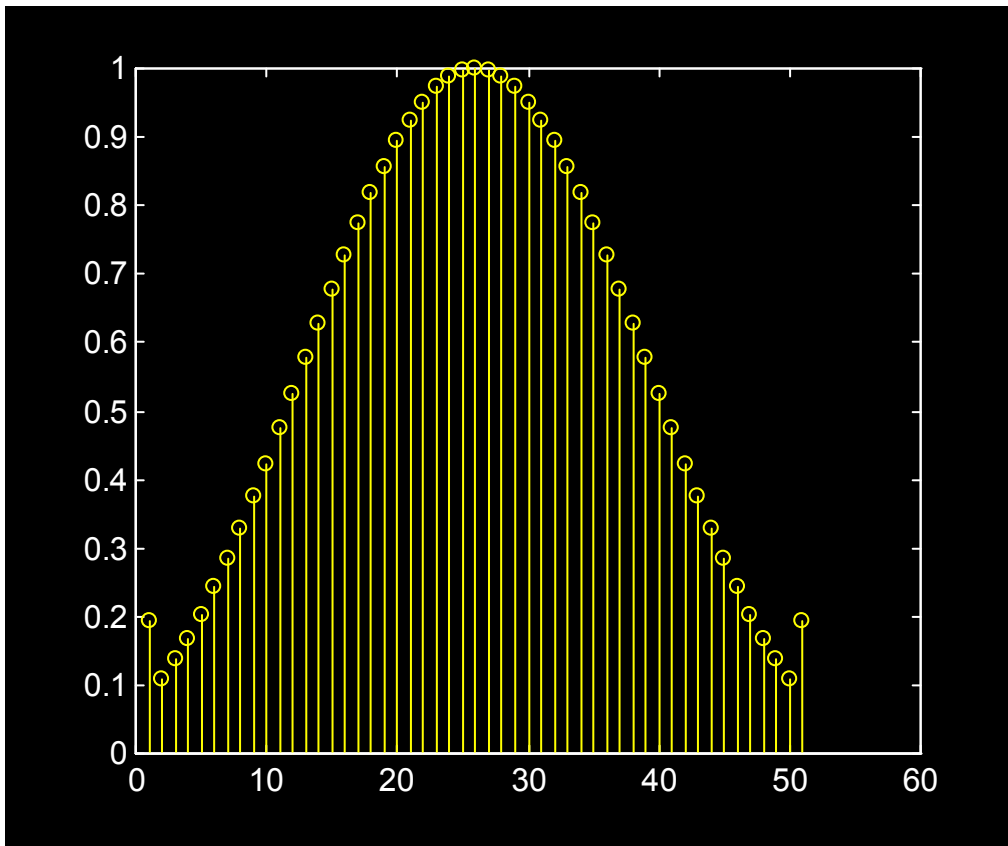
Plot 11 Chebyshev response



The Chebyshev window is shown in Plot 12. As shown in the plot the Chebyshev window has large spikes at its output samples. Also Chebyshev windows can only be defined for odd lengths. Note the frequency response with exactly 40 dB side lobe height.

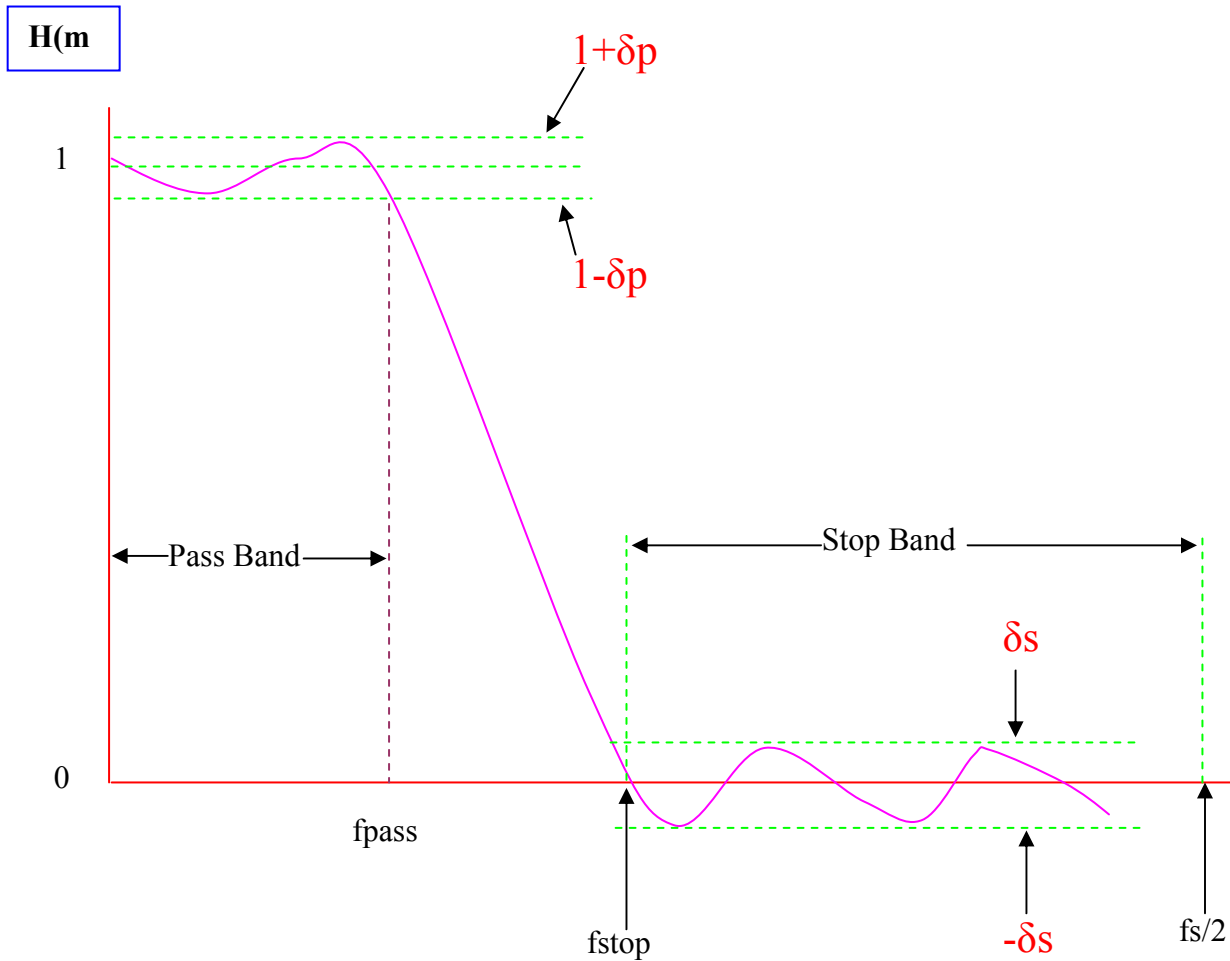
The Chebyshev window's parameter γ allows variations in the response to be made. Most CAD programs accept this parameter or the maximum height of the side lobe levels with respect to the main lobe. The stop band attenuation for the Chebyshev window is 20γ . For example if we want the side lobe levels to be no greater than -60 dB below the main lobe, we would pick $\gamma = 3$.

Plot 12 The Chebyshev window for the above design



The Remez Exchange Method (or Parks – McClellan or Optimal method):

This technique is a popular technique used for the design of high performance FIR filters. Given a desired filter response as shown below:



Plot 14 Specifying the Low Pass Filter

Here we establish a pass band cutoff frequency, a stop band start frequency , f_{pass} and f_{stop} . The quantities δ_p and δ_s are the ripple in the pass band and the stop band. In terms of dB's these are:

$$\text{Pass band ripple} = 20\log_{10}(1 + \delta_p)$$

$$\text{Stop band ripple} = -20\log_{10}(\delta_s)$$

Using a Remez CAD routine we can find the N coefficients. An example is shown below:

[Sometimes filters designed using the Remez Exchange Method are also called equi-ripple filters because of the way they behave with respect to ripple.]

The CAD routine being used in this example (as in the above) is the Signal Processing ToolBox from MathWorks used with MATLAB.

The routine $b = \text{remez}(n, f, m)$ returns a row vector b containing the $n + 1$ coefficients of an order n FIR filter whose frequency magnitude characteristics match those given by vectors f and m .

- f is a vector of pairs of frequency points, specified in a range from 0 to 1, where 1 corresponds to the sampling frequency (Nyquist frequency). These frequencies must be increasing order.
- m contains the magnitude points at the frequency points specified in f .

For example:

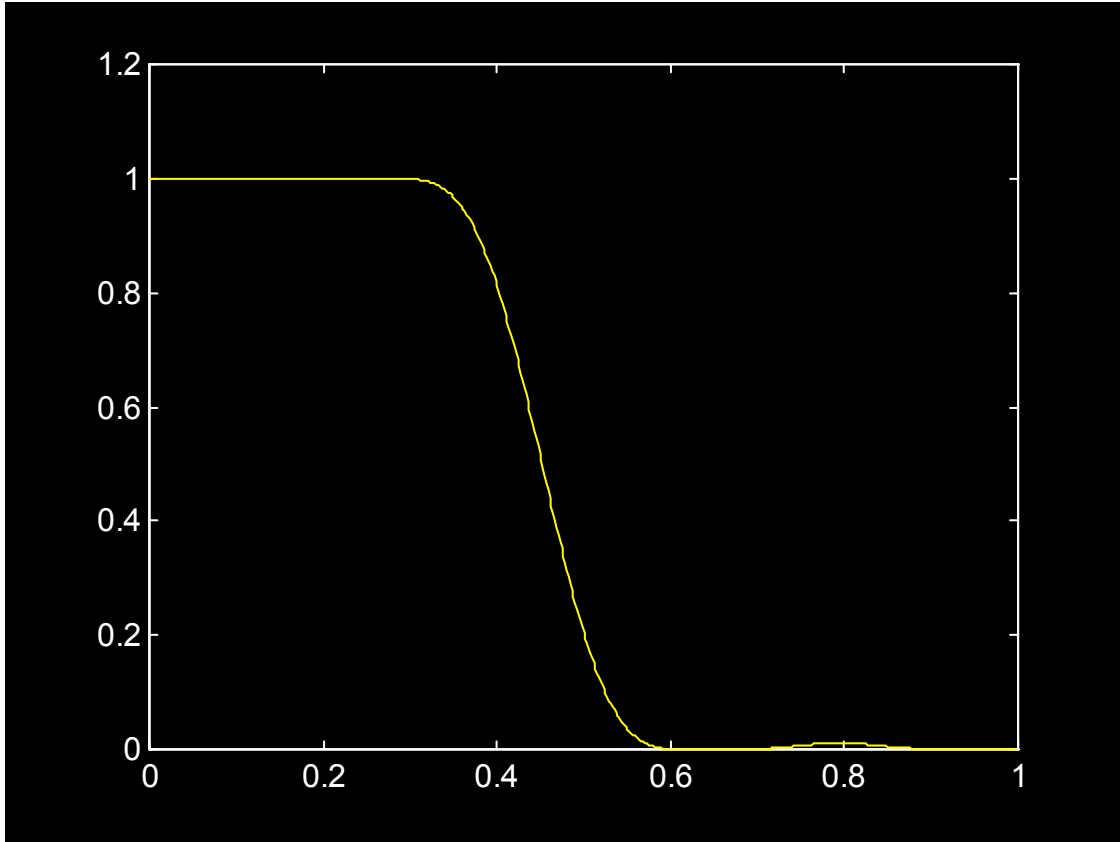
TABLE R1.0 For Remez FIR design example

f	m
0	1
0.1	1
0.2	1
0.3	1
0.6	0
0.7	0
0.9	0
1	0

The frequency points are in pairs with corresponding magnitude values. Obviously the specification above is for a low pass filter. The Remez design method gives the designer a Chebyshev type filter whose actual frequency bresponse is as close to the desired response for a given number of taps. The response shown below in Plot R1. was made using 25 taps.

This Remez method assumes that we want the ripple to be as small as possible. Remez design methods usually provide the steepest slopes whiel keeping the ripple as low as possible.

Plot R1. Remez Exchange method filter response.



Half Band Filters:

There is another type of FIR filter that is worth mentioning here. This is the [half band filter](#). The frequency response of this type of filter is symmetrical about the $(1/4)fs$ point on the frequency axis. Because of this characteristic every other tap value of this filter is equal to zero. This implies that we do not need to do a multiplication for that tap reducing the need for multiplications!! A very cost effective and useful property indeed since multiplication takes time and silicon area. The fewer the multiplications the lower the cost. Half band filters find use in *decimation* operations.

Other filter types:

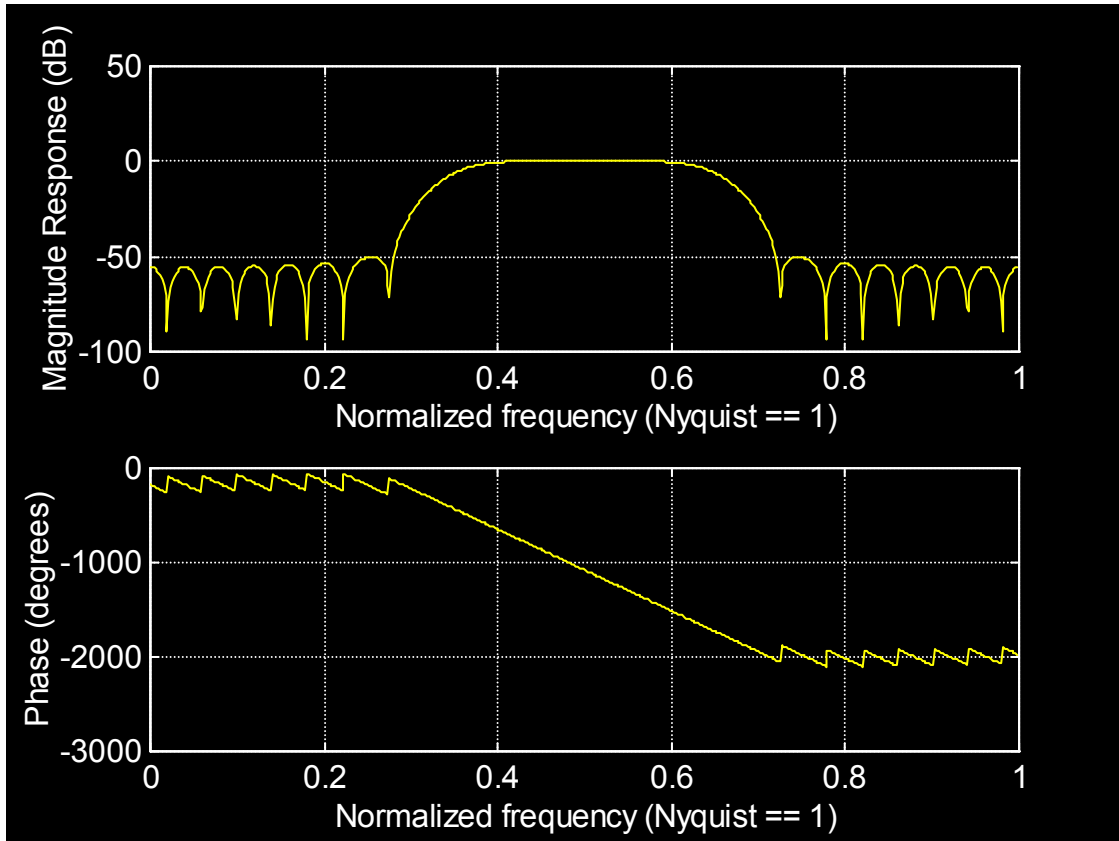
So far the discussion has been focused on low pass filters. The design of band-pass, high pass and band-stop filters is similar. Use is made of available CAD programs to generate the values of the FIR coefficients.

In the following, the design of these types of filters is provided by example. CAD software was used freely. Links and references to this software will be found at the end of this section.

Band – pass filters:

The response below (Plot BP1) is for a band-pass filter. The pass band lies between 0.35 and 0.65 fs.

Plot BP1. Band- pass FIR Filter using a Hamming Window design.



The number of coefficients is 48. The impulse response is shown below in Table BP1.

Table BP1. Impulse response for the band – pass filter.**Tap Values**

-0.0020
0.0000
0.0022
0.0000
0.0000
0.0000
-0.0062
0.0000
0.0118
0.0000
-0.0059
0.0000
-0.0169
0.0000
0.0421
0.0000
-0.0361
0.0000
-0.0284
0.0000
0.1424
0.0000
-0.2542
0.0000
0.3008
0.0000
-0.2542
0.0000
0.1424
0.0000
-0.0284
0.0000
-0.0361
0.0000
0.0421
0.0000
-0.0169
0.0000
-0.0059
0.0000
0.0118
0.0000
-0.0062
0.0000
0.0000
0.0000
0.0022
0.0000
-0.0020

The design process for the other filter types is the same. Using appropriate software specify the filter, then by manipulating the results obtain the tap coefficients. Once the tap coefficients have been obtained the filter has been designed.

Phase response of FIR filters and group delay considerations

A dominant feature of the FIR filter is its linear phase response. The response of the FIR filter can be described by a magnitude function $H(m)$ which consists of a real part and an imaginary part.

The phase of $H(m)$ is the arctangent of the imaginary part divided by the real part. Looking at Plot BP1 the phase of the designed filter has a linear phase response in the pass band.

This is indeed why FIR filters are preferred in many applications. The question is why care about linear phase at all?

To appreciate this property of FIR filters the concept of group delay must be introduced. Group delay is the rate of change of phase with respect to frequency. If phase is ϕ and frequency is f then:

$$\text{Group delay} = \Delta \phi / \Delta f$$

When group delay is constant, as it is in FIR filters (within the pass band), all frequency components of the input signal are delayed by an equal amount of time. This means that no phase distortion is introduced by the filter itself.

This property is crucial in some systems like communications, some video systems etc.

Group delay is sometimes also called envelope delay.

We are not concerned with group delay outside the pass band since we are actually trying to eliminate energy outside the pass band by filtering.

For a N tap FIR filter, group delay can be calculated by:

$$\underline{GD_{\text{odd}} = (N-1)T/2 \text{ for } N \text{ odd and } GD_{\text{even}} = NT/2 \text{ for } N \text{ even}}$$

$$\underline{\text{Where } T = \text{sampling period.}}$$

To summarize: The phase at the output of a FIR filter is the phase of the first output sample relative to the phase of the first input sample. The phase shift is linear function of frequency in the pass band. However, this will be true only if the filter has symmetrical coefficients.

There are a few other issues that require addressing. These are described in the following appendices.

- (A) [The effects of sampling and generation of periodic responses](#)
- (B) [The effects of fixed point binary word lengths](#)
- (C) [Binary number formats and their uses](#)
- (D) [Links and references to software packages used in this discussion](#)

[Appendix A The effects of sampling and generation of periodic responses](#)

When a sine wave signal at frequency f_0 , is sampled with a periodic waveform at frequency f_s , the samples obey the rule:

$$x(n) = \sin(2\pi(f_0 + k f_s) n t_s)$$

[where n = sample number, k = index \(\$\pm\infty\$ \) \$t_s\$ =sampling time](#)

The important implication of this equation is that an $x(n)$ sequence of digital numbers (samples), representing a sine wave at f_0 , also represents sine waves at other frequencies, $f_0 + k f_s$.

[In other words, when sampling at a rate of \$f_s\$ samples/sec, if k is any positive or negative integer, we cannot distinguish between the sampled values of a sinewave of \$f_0\$ Hz from a sinewave of \$f_0 + k f_s\$ Hz.](#)

Accordingly, **no** sequence of values stored in a computer, for example, can unambiguously represent one and only one sinusoid without [additional information](#)! A sampled sequence represents an infinite number of different sinusoids.

[This is a fundamental relationship in digital signal processing.](#)

When the input is a band of frequencies, the same effects occur and the same rule applies. Consider FIGURE A1. In this figure a baseband signal limited in bandwidth between $\pm B$ is sampled at a rate f_s . The resulting output signal in the frequency domain is shown in the same figure. Note the replications of the input signal at multiples of the sampling frequency f_s . This is a consequence of the sampling operation.

In a practical sampling scheme, the sampling frequency is always greater than $2B$ to avoid the overlap in frequency of the replications of the input as shown in the figure. This is known as the Nyquist criterion and the frequency is known as the Nyquist frequency.

If the Nyquist criterion is not obeyed corruption of the sampled output will occur and parts of the signal will be lost. Another effect is that [all the information in the original signal now lies within the band from \$-f_s/2\$ to \$+f_s/2\$](#) . This warns us that any signal located beyond $+B$ or below $-B$ [will always end up in the band of interest](#). Any noise or other

unwanted signal beyond the band limits should always be attenuated using an analog continuous time pre-filter.

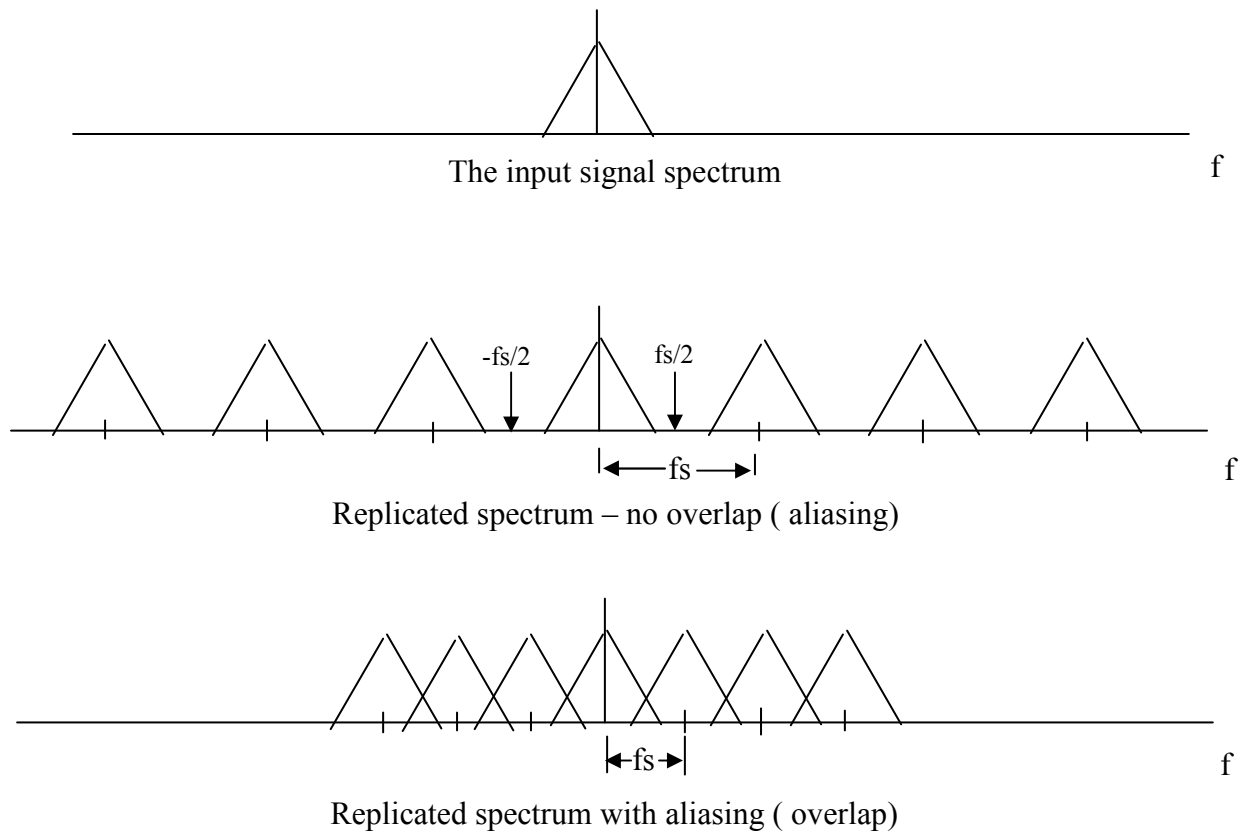


FIGURE A1

Obviously the original information may be extracted from the sampled signal by bandpass (or low pass) filtering the resulting sampled output.

[Appendix B: The effect of fixed point binary word lengths](#)

In this appendix we explore what happens when we do a A/D conversion of a continuous signal or design a FIR filter with finite word length digital samples.

It should be apparent that the number of bits used to represent a number is a critical parameter. The more bits used, the better the resolution and vice versa. If we assume that the binary word is used to represent the amplitude of the signal, then its dynamic range is given by:

$$\text{DR} = 6.02 \cdot b \text{ dB where } b = \text{number of bits.}$$

For a 12 bit word, DR = 72.24 dB.

When a finite word length is used, it prevents the representation of a number to an infinite precision, increases background noise (quantization and otherwise), generates filter responses that may be non-ideal and may lead to severe mathematical errors.

The primary errors are:

A/D conversion errors

Data overflow

Truncation

Rounding

A/D Conversion: It can be shown that the SNR of an A/D converter is given by the expression:

$$\text{SNR(A/D)} = 10 \cdot \text{Log}_{10} \left\{ \frac{[(\text{LF})^2 \cdot \text{Vp}^2]}{[\text{Vp}^2 / (3 \cdot 2^{2b})]} \right\}$$

Where,

LF = loading factor = rms value of input signal/Vp

Vp = maximum input peak voltage of the A/D

b = number of bits of A/D resolution

Any continuous signal that is digitized by the A/D will have at most a SNR represented by the above equation. Therefore choice of the number of bits of conversion is important. In addition there are other sources of noise in an A/D which can further contaminate the signal, such as: aperture jitter, missing output bits, non linearities etc. All of these must be taken into account when choosing the right A/D converter otherwise there will be error in the output of the filter.

Note that the expression of A/D SNR is given under the assumption that the input to the A/D is driven to its peak. However, this can be dangerous as distortion may result if there is any overdrive at all. The expression above provides an optimistic number for A/D SNR.

Choosing the right A/D is largely based on engineering judgment based on considerations such as the above. When in doubt choose a larger resolution A/D!

Data overflow:

Data overflow is what happens when the results of operation results in a number of bits that cannot be held by the registers of the digital machine. In this case the usual technique is to discard the LSB (least significant bit). Generally this may not be a problem, however in some cases it could lead to numerical noise which will cause problems in filter SNR and inaccuracies in the filter characteristics.

The general rule is that the sum of M individual b – bit binary numbers can require as many as $[b + \log_2(M)]$ bits to represent the results.

Generally the use of 2's complement binary formats ease the problem of data overflow.

Truncation:

The process of truncation represents a data value by the largest quantization level that is less than or equal to that data value. This should be taken into account when designing the digital machines used for the various function blocks (multipliers, adders) in the FIR filter. The filter design should also be tested rigorously under the truncation error regime to make sure of the accuracy needed for that particular filter.

Rounding:

A data overflow error can be rounded off. I.e. round off a particular data value to its nearest quantization level. Again it is obvious that a loss of accuracy results and it introduces noise. However, it can be shown that the error introduced by rounding is usually less than truncation. So rounding may be an alternative to truncation in some applications.

All of these errors using fixed point binary formats occur because of the requirements in filtering for multiplications and additions. In hardware implementations of FIR filters (that is the subject of this e – book) this challenge can be met by the use of floating point binary numbers.

[Appendix C: Binary number formats and their uses](#)

The formats to be discussed in this appendix are:

Fixed point binary
Octal
Hexadecimal
Fractional binary
Sign – magnitude binary
Twos complement
Offset binary
Floating point binary

Fixed point binary:

This is the simplest format of all. Numbers are represented by a 1 or a 0. The left most bit is the MSB (or Most Significant Bit) and the right most bit is the LSB (or Least Significant Bit). The base of the system is 2. The positional value of each bit is the base

raised to power of the position multiplied by the bit (1 or 0) as shown below in the example.

$$\underline{\text{Decimal } 5 = \text{Binary fixed point } (101) = 2^2 \cdot 1 + 2^1 \cdot 0 + 2^0 \cdot 1}$$

Octal:

The base in the octal number representation is 8. The digits in the octal number system are 0, 1, 2, 3, 4, 5, 6, 7. The same positional values are used. For example:

$$\underline{\text{Decimal } 64 = \text{Octal } (100) = 8^2 \cdot 1 + 8^1 \cdot 0 + 8^0 \cdot 0}$$

To convert to/from binary to octal simply group the binary bits into groups of 3 and then replace by the binary value of the groups of three. For example:

$$\text{Binary } (10\ 101\ 001) = \text{Octal } (251)$$

Hexadecimal (or Hex for short):

The base in Hex is 16. The digits in the hexadecimal system are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. The same positional values are used. For example:

$$\underline{\text{Decimal } 1026 = \text{Hex } (3F2) = 16^2 \cdot 3 + 16^1 \cdot 16 + 16^0 \cdot 2}$$

To convert to/from binary to hex simply group the binary bits into groups of 4 and replace by the binary values as shown below:

$$\text{Binary } (1010\ 1001) = \text{Hex } (A9)$$

Fractional binary:

Fractional binary numbers are those that have values between 0 and 1. Fractional binary numbers use a binary point. For example a 6 bit binary fractional number is:

$$\underline{\text{Decimal } 3.3125 = (1.2) + (1.1) + (0.1/2) + (0.1/8) + (1.1/16)}$$

Sign – magnitude binary format:

To represent positive and negative numbers a signed magnitude binary number uses the most significant bit as a sign bit. For example:

$$\underline{\text{Decimal } -25 = 111001 \quad \text{Decimal } 25 = 011001}$$

Two's complement format:

Two's complement also uses the MSB (or the left most bit) as a sign bit. This format is a very popular format for hardware implementations. The reason is that the same hardware can be used to do addition and subtraction.

In a two's complement format all that is necessary to change a positive number to a negative number is to change all ones to zero's and all zero's to one's. and add a one to the complemented word.

For example:

Decimal 3 = 0011 to Decimal -3 = 1101 is generated by the following steps:

0011 → 1100 → add one → 1101

In two's complement a b – bit word can represent positive magnitudes as large as $2^{b-1}-1$ And negative magnitudes as large as -2^{b-1} .

There is a pitfall to using two's complement numbers. Adding two numbers of different word lengths can be tricky. For example consider the subtraction of 3 from 12 or

0001100		▶ 12
101		▶ 3 in two's complement
0010001		▶ Incorrect

The solution that is used is to sign extend the two's complement of 3 as shown below:

```
0001100
1111101
```

0001001 (discarding the left most overflow) This is the correct answer!

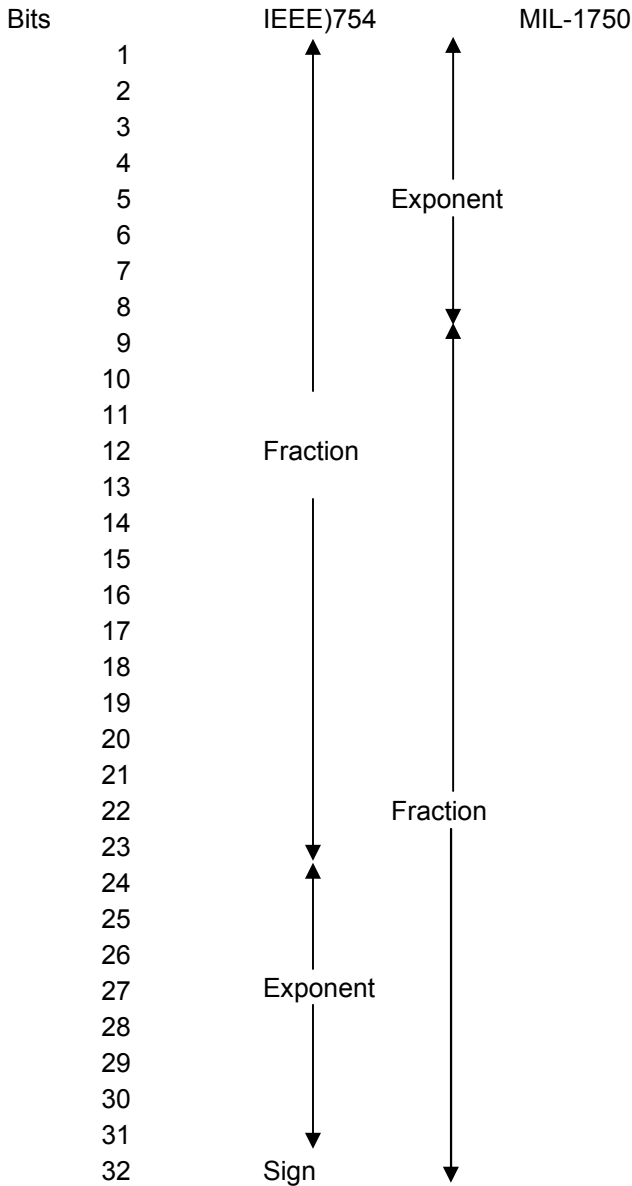
Floating point binary format:

The floating point number format partitions a binary word into a mantissa and an exponent as shown below:

$$N = m \cdot 2^e$$

The mantissa and the exponent can be positive or negative. The more bits that are used for the exponent the larger the number will be. The more bits used for the mantissa the greater the resolution or precision.

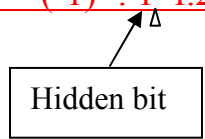
Two popular floating point formats are the IEEE P754 format and the MIL – STD 1750A format (in case of military designs). The features of both formats are shown below:



Note: The IEEE format is the most popular since so many manufacturers of floating point circuits use it. The exponent is offset binary and its fraction is a sign magnitude binary number with a hidden bit that is assumed to be 2^0 .

To evaluate the decimal value of a IEEE floating point number use the formula below:

Decimal value = $(-1)^s \cdot 1.f \cdot 2^{e-127}$ where e is the exponent. Δ is the binary point.



On the other hand the MIL – 1750 is a US Military Airborne floating point standard. Its exponent e is a two's complement binary number residing in the LSB bits. The fraction is also a two's complement number with no hidden bit. That is why a sign bit is not indicated above. The decimal value of the MIL – 1750 format can be calculated by using the formula below:

$$\text{Decimal} = f \cdot 2^e$$

Specific information about number formats is of course prolific in the literature so will not be addressed here. The purpose of this appendix is to make the reader aware of the various issues concerning number formats.

[Appendix D: Software used.](#)

A majority of the CAD work done in this section of the e – book used MATLAB. This package can be obtained from the MathWorks. www.mathworks.com
The specific toolbox that was used is the Signal Processing Toolbox.

Signal Processing Group Inc., offers extremely cost-effective services for the design, development and manufacture of analog and wireless ASICs and modules using state of the art semiconductor, PCB and packaging technologies. For a completely no-obligation quotation please send us your requirements. Email : spg@signalpro.biz.