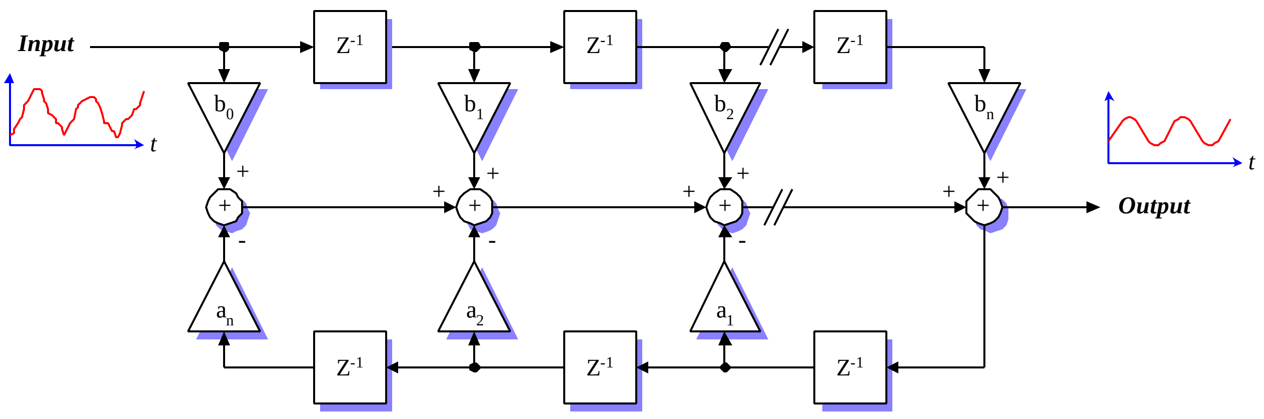
Elite Engineering Corp.
Robotics and Instrumentation Systems

Digital Filters - A Basic Primer



Written By:
Robert L. Kay
President/CEO
Elite Engineering Corp

Notice!

Introduction

Since the advent of inexpensive processing power, the general engineering community has had opportunity to apply digital filtering techniques to signal processing applications. Filter functions that were not realizable in analog implementations came into practicality. Filters with very steep cutoffs or very narrow notches could be realized by software implementations. An amazing amount of signal to noise improvement became attainable with systems having spectrally separated and unique signal components. As digital filtering became more and more popular, unique integrated circuit chipsets and processors were developed to provide high speed, compact implementations. Complementary design tools were developed to speed development time and simplify the necessary software coding requirements.

However, this new tool still requires an understanding and some expertise in mathematics, most notably of Fourier and Laplace transforms. The progress in personal computer software tools has reduced the need to manipulate the mathematics symbolically. This makes application of software driven digital filters a straightforward number crunching task in most cases. Yet it is still crucial for the designer to understand the principles at work. The designer should know the basic principles behind filters, FFT=s and numerical processes.

Medical products require a broad range of signal processing requirements. Generally speaking, medical analyzers present some of the most interesting and rigorous design challenges in the signal processing arena. Most diagnostic devices are based upon instrumentation of one form or another. In the development of the majority of that type of instrumentation, the goal is to maximize signal to noise ratio. Interference from AC power, motors, and such can dramatically reduce the ability of low cost equipment to achieve its best performance. Digital filtering can provide a low cost solution to reducing noise beyond what can be achieved with typical analog solutions. This article provides the reader with a basic introduction to digital filters and provides a simple tool to allow the reader to explore these fascinating tools in greater detail.

Basics

In general, the need for filtering arises when the signal of interest is mixed with interfering signals or noise. Almost all noise or sources of interference can be analyzed from the frequency domain perspective. In order to determine which filter type and characteristics are required for any particular application, the spectral characteristics of the signal and its noise (or interfering components) is necessary. The designer must then define the required output SNR (signal to noise ratio), response time, overshoot and other parameters as may be deemed significant to the application. When that is complete, the filtering requirements can be established and a detailed design can commence.

In the past, analog filters were the primary means to performing the necessary filtering to separate the signal from the noise components. In some cases, the design of multi-pole analog filters required a number of relatively large components to achieve the required degree of filtering. The resulting filter was susceptible to minor variations in component values and often required additional amplifiers to perform buffering. The stability of these amplifiers required a reasonable degree of analysis which might be offset using cookbook designs and simulation software. Sharp filter functions were expensive and time consuming to implement. Predicting the response of these filters to all variations and scenarios was difficult and time consuming. Amplifiers saturated, inductors saturated, and capacitors exhibited dielectric absorption to name a few of the ill=s that analog designers had to deal with. The reader must be fully aware that with digital filters, the basic concepts behind these problems have not changed and they now plague the digital engineer too.

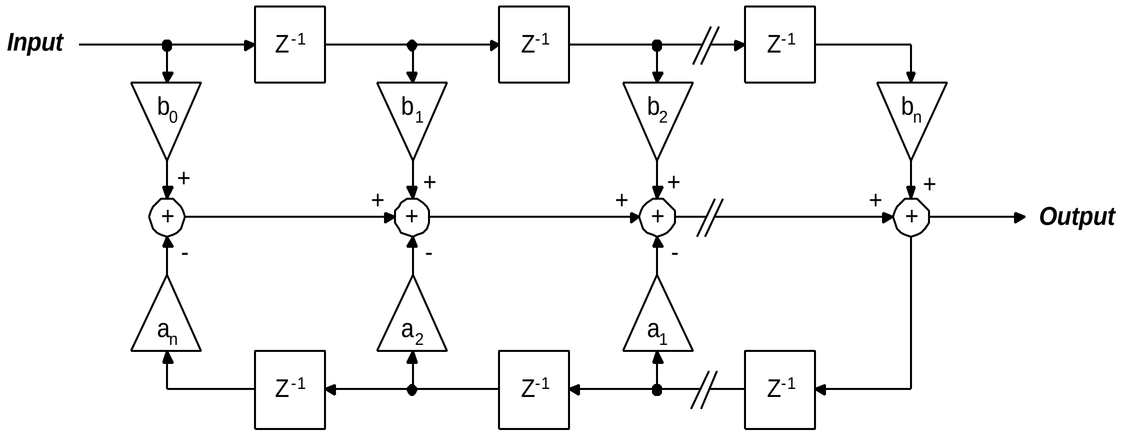
Digital Filters - General Form

Figure 1 shows the basic structure for a general purpose digital filter. The structure is composed of Z-domain delay blocks, gain stages and summing junctions. Observation of this structure shows how the filter processing each input sample as well as the output. Each time the system samples the input, the top half of the system takes previous input samples and shifts them down the chain of Z^{-1} blocks. Thus the system Aremembers@ a fixed series of past input values. Each past value and the present input value is multiplied by a fixed coefficient. These terms are summed up to create the filter output.

The lower half of the diagram shows how the filter can also use the current output value as well as Aremember@ past output values. These values are also multiplied by fixed coefficients and summed to add into the current output value.

In general, each Z^{-1} element in the filter structure creates a pole in the response. As noted in the figure, an FIR (Finite Impulse Response) filter will have some or all A*b*@ terms but will not use any A*a*@ terms. An IIR (Infinite Impulse Response) filter will have A*a*@ terms but may not have some of the A*b*@ terms (it must have at least one, usually b_0). There can be much debate regarding the appropriateness of each type of filter. For the purposes of this article, it is more important to notice that an FIR filter uses memory to recall past inputs and integrate them with a specific weighting function (i.e. the A*b*@ terms). Thus an FIR uses no feedback, is inherently stable and will not oscillate. On the other hand, an IIR filter provides feedback via non-zero values for the Aa@ terms (*b* terms may or may not be present). This allows the potential for oscillation and thus such filters are not inherently stable.

General Purpose Digital Filter Configuration



Z^{-1} = Unit Time delay (sampling delay)
a_x, b_x = simple gain blocks

For an FIR filter, all a_x terms are zero.
For an IIR filter, some or all b_x terms may be zero.

The choice of an FIR design versus an IIR can spark much debate among designers. It is the opinion of the author that new users of digital filters should start out with FIR type filters. This is recommend for the following reasons:

$ FIR filters can easily be designed for constant phase delay and/or constant group delay (which affects the distortion of pass band signals with broadband characteristics)

$ Stability is inherent and limit cycling is not a problem as it is with IIR designs (This is provided the User implements the filter with nonrecursive techniques).

$ Round off errors can be controlled in a straightforward fashion in order to keep their effects insignificant.

The drawbacks of using an FIR are:

$ An FIR generally requires more stages than an IIR to obtain sharp filter bands.

$ Additional stages add to memory requirements and slow processing speed.

Designs that demand high performance usually justify an effort to tradeoff FIR vs IIR filter implementations. But for first time application it is recommended that aggressive filter design be avoided until one has the experience to avoid the various pitfalls that await you.

Mathematically, the equation representing the general filter in Figure 1 is:

$$y_i = \sum_{n=0}^{m} b_n * x_n + \sum_{n=1}^{m} a_n * y_n \qquad \text{Equation \#1}$$

Where: x_0 = current input value
y_0 = current output value
m = number of filter stages or taps
(Z^{-1} sections)

Equation #1 is straightforward to implement in software or even a speadsheet. As an example, an FIR filter (lowpass, high pass or bandpass) would be constructed as follows:

1. Determine how many poles are needed and how much ripple in the pass band is allowed and how much attenuation in the stop band(s) is required.

2. From this information, compute the coefficients necessary.

3. Setup your software as follows:
   a. Create an array to hold the incoming input values as a sequence of inputs. Create a matching array to hold the coefficients.

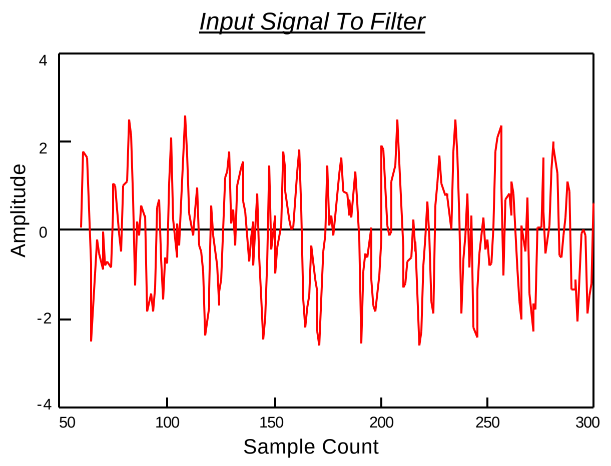
   b. Set up a software mechanism to add incoming input values to the Ax@ array. A circular buffer works well for this purpose.

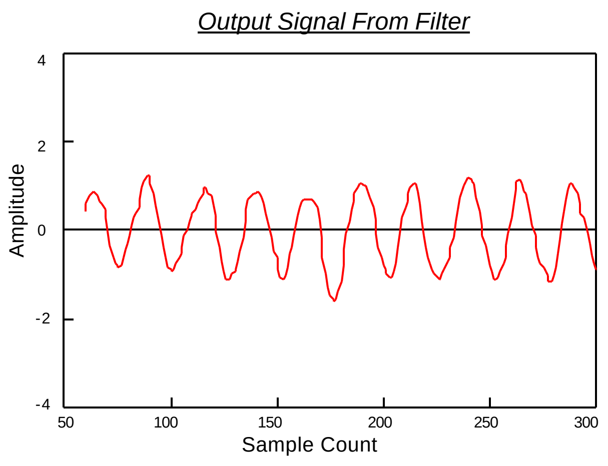
   c. Set up a function to multiply the current input array by the matching array of coefficients and to then sum the result. That result is the output of your filter.

Whether you are performing real-time processing or off line processing, the process is essentially the same. Using a circular buffer is an important means to saving computational time. Without some sort of circular buffer, you must shift the array of input values on each input or you must shift the coefficients. This shifting can be time consuming and may be time reduced dramatically by a circular buffer technique.

Figure 2 shows an example lowpass filter. The filter was designed using the Remez Exchange Algorithm to generate the coefficients (explanation of this algorithm is given the following paragraphs). The input signal to the filter was created by combining two sine waves (one near the edge of the passband and one just outside the pass band). Random noise was added to the sine waves. The noise added is uniform in spectrum and thus a portion of the noise power overlaps the passband signals. Thus the filter output still shows some low frequency components riding on the sine wave. This noise could be further reduced by use of a bandpass configuration as would be appropriate in a strictly analog implementation.

Lowpass Filter Example





Input Signal Composition

1 Sinewave at 0.04 Fs (Fs = Sample Frequency)
1 Sinewave at 0.15 Fs
Random noise, uniform distribution,
(rms amplitude = 1)

Filter Characteristics

Passband Ends at = 0.05 Fs
Stopband Starts at = 0.15 Fs
Stopband Attenuation = 30bD (min)
Passband Ripple = 0.1dB (max)
17 Terms or taps

It should be apparent at this point that the whole key to these filters is understanding how to generate the coefficients. Implementing the filter itself becomes a small task compared to the mathematics to generate the coefficients.

Computing The Coefficients

Generating the coefficients for these filters is far and away the toughest part of filter design from a mathematical standpoint. The complexity of generating these coefficients is driven by the criteria for obtaining specific pass band, stop band and ripple requirements. While a number of software programs are available to reduce this process to cookbook level, it is important to understand the principles and processes at work. A general observation of the filter structure shown in Figure 1 should bring to mind the basic convolution process. If one views the filter coefficients as a reverse ordering of the filters impulse response, then it becomes obvious that these filters are simply a discrete embodiment of the convolution of the filters impulse response against the incoming time domain signal. Thus one can view the algorithms for generating the coefficients simply as tools that generate an array of impulse response values whose Fourier Transform represents the desired filter function.

A derivation or detailed explanation of the mathematics for the various approaches will not be presented here. The reader is encouraged to review the text in reference [2] for specific detail and description of the mathematics for the described approaches.

Three popular basic approaches for coefficient generation are employed by filter designers: the Fourier Series Method, the Frequency Sampling Method, and the Remez Exchange Method. The Fourier Series Method is based upon direct computation of Fourier series coefficients given a selection of the desired start and stop frequencies. These coefficients are computed using the following equation:

$$b_n = (1/(2p)) \int_{2p} K(f)[\cos(mf) + j\sin(mf)]\, df \qquad \text{Equation \#2}$$

Where: n = index of coefficient being computed
f = the frequency currently being computed
K(*f*) = filter gain at frequency *f*
m = n - (N-1)/2
N = number of filter stages or taps

Note that m is setup to compute only half the number of coefficients as there are taps. This is because the coefficients will be symmetric and will generate a symmetrical impulse response. For example, a 10 tap system will have the following arrangement:

$b_0 = b_9$
$b_1 = b_8$
$b_2 = b_7$
$b_3 = b_6$
$b_4 = b_5$

If an odd number of taps are specified then the system will be symmetric around the center most tap. For an 11 tap system this would mean $b_0 = b_{10}$, and b_5 would be the only uniquely assigned value.

The basic steps to compute the coefficients, b_n, are as follows:

1. Specify the desired filter attenuation at each frequency, i.e. the K(f) coefficients.
2. Specify the number of taps or stages, N.
3. Compute each coefficient per equation #1.

Some iteration may be required to determine the appropriate number of taps based upon the resulting filter step response and/or its frequency response (obtained via an FFT or its impulse response).

Filters designed using Fourier Series Method tend to have linear phase characteristics. As with almost all filter implementations, the filter will exhibit undershoot and overshoot due to the Gibbs phenomena. This arises due to the finite structure of the digital computations as opposed to a continuous system. The impact of this effect may be reduced by a technique called Awindowing@. This involves multiplying the coefficients by a series of coefficients designed to modify the frequency response in order to adjust for the finite aspect of the transforms taking place. While it is beyond the scope of this article to detail out these functions, it is recommended that the reference material be reviewed for information relating to the various window types: rectangular, triangle, Hanning, and Hamming. The primary tradeoff between each windowing technique is whether or not the application requires smoother frequency response for phase or magnitude.

Another technique used to generate coefficients is to utilize a discrete Fourier Transform. In this technique, the designer generates points that represent the desired frequency domain magnitude and phase response desired. The resulting data set is processed via a standard inverse discrete Fourier Transform. This produces impulse response values similar to the Series Method. These values are utilized in the same manner. The primary difference is that the designer must (or can) specify magnitude and phase in this technique. This technique is not recommended for those not familiar with Fourier Transforms and the effects of magnitude and phase point placement on the resulting transform. The choice of point placement can be quite important in this method and is one that can produce significant changes in results from subtle changes on the inputs.

The third technique is called the Remez Exchange Method. This algorithm utilizes inputs from the designer for start/stop frequency, allowed passband ripple, and minimum required stop band attenuation. The algorithm then attempts to find an optimal solution of coefficients to meet the given criteria. The mathematics of this algorithm are explained in a reasonably clear fashion in reference [2] and will not be covered here. A number of software packages are available that perform the necessary computations which make this approach attractive as an almost Acookbook@ solution to generation of coefficients. Depending upon the specific implementation of this algorithm, the usual criteria coded into the algorithm produces Chebyshev filter approximations.

Additional Functions That Affect The Digital Filter

A digital filter is only one part of a system composed of a digitizer (A/D), processing device (e.g. microprocessor) and algorithm. The digitizer has a profound effect on the filtering function via three primary characteristics: quantizing error, sample rate and bandwidth limiting. Quantizing error is simply the number of bits the A/D uses for quantization. Determining the required number of bits is usually governed by the system resolution requirements (i.e. a 2.4mV change in the signal is required to be detectable). With regard to the digital filter, it is difficult to remove quantization noise without undue effort or a priori and fixed frequency characteristics of the signal and noise. Thus quantization size must be chosen to be compatible with the desired filter output SNR.

The sample rate of the A/D is a key parameter for a digital filter. The basis of setting the coefficients for almost all digital filters is an assumption of the sample rate. Changing the sample rate will directly scale the coefficients. A filter you thought filtered at 60Hz will not do so if the sample rate changes. This is a one to one relationship. Thus, you can determine the effects of varying sample rate by the ratio of change from the design specification frequency to the actual rate. The resulting filter frequency is scaled by the same ratio.

Jitter or short term variation of the sample frequency can create very undesirable effects on the filter performance. Simply think of it as the filter breakpoint frequencies shifting around in real time. This can be a significant problem for narrowband or sharp cutoff filters where the signal of interest lies very close if not within these filter bands.

The anti-alias filter is perhaps the most crucial addition to the input of the A/D and to the digital filter. Without an analog filter to perform this function, high frequency interference can show up in the digitized data as a low frequency signal. Those familiar with sampling theory will know this as frequency folding or aliasing. Once aliasing occurs, it is very difficult if not impossible to remove the high frequency components that folded back into the sampled bandwidth. By adding an analog filter that limits the A/D input bandwidth to one half the sample rate, aliasing can be reduced or prevented. Most applications use a simple analog RC filter to perform this function. The reader should be aware that this filter does not roll off very quickly and thus large amplitude interference above the cutoff frequency can and will still enter the system and be folded back into the sampling bandwidth. Higher order analog filters are still required where this situation is possible to occur.

The processor chosen for the task plays an important role in a number of ways. The most obvious is the processor=s math function capabilities. Digital filters inherently are composed of repetitive multiplication and addition operations. More over, round off errors can accumulate quickly. Hence, most software applications utilize floating point variables to reduce round off error accumulation (some require double precision for high accuracy applications). Floating point multiplication’s generally consume large quantities of processor time unless a math coprocessor is present. This can prevent the processor from performing real-time processing if it interferes with maintaining a uniform sample rate. Thus processor speed and capability are important parameters in the implementation of a digital filter.

Summary

Digital filters can be a valuable tool in an engineer=s tool box. Properly applied, SNR improvements can be achieved that surpass those attained by analog techniques. Yet one can not simply throw a digital solution in place without understanding the underlying principles and concepts. Thus if digital engineers are to utilize these techniques, they had better be prepared to become versed in the mathematics that used to be reserved for analog engineers.

As a courtesy to those readers interested, a Acookbook@ software package to compute filter coefficients using the Remez Exchange Algorithm is available, free of charge from the author.

References

[1] Digital Filter Designers Handbook, C. Britton Rorabaugh, McGraw Hill, 1993
[2] C Language Algorithms For Digital Signal Processing, Paul Embree, Bruce Kimble, Prentice Hall, 1991