ANALOG DEVICES

AN-344
APPLICATION NOTE

ONE TECHNOLOGY WAY • P.O. BOX 9106 • NORWOOD, MASSACHUSETTS 02062-9106 • 617/329-4700

Digital Fir Filters Without Tears

by Bill Windsor and Paul Toldalagi

Digital filters once required specialized design techniques, high-performance costly hardware, and complicated software to implement; for these reasons, they found only restricted application. Today, in sharp contrast, the availability of low-cost high-speed digital signal-processing ICs, such as multipliers and multiplier/accumulators, combined with easy-to-use standardized design procedures, has dramatically simplified filter implementation. Consequently, if your applications require filters with rolloffs in excess of 24 dB/octave, you should include digital filters in your design repertoire.

In these pages, we will compare digital and analog filters, discuss the various digital filter architectures, and—as an example—show you a step-by-step method of designing FIR (non-recursive) filters. A set of references will show you where to find information on topics only touched on lightly here.

COMPARE DIGITAL AND ANALOG FILTERS

Digital filters increasingly find their way into modems, radars, spectrum analyzers, and speech- and image-processing equipment, and for good reasons: Compared to analog filters, digital designs offer sharper rolloffs, require no calibration, and have greater stability with time, temperature, and power-supply variations. Simple software changes can alter a digital filter's response in real time, creating so-called "adaptive filters," whereas analog filters usually require hardware changes.

But digital filters do not satisfy every application. Analog techniques are usually most cost-effective in designs calling for rolloffs of up to about 24 dB/octave. As rolloff requirements exceed 24 to 36 dB/octave, however, digital filters increasingly make more sense. In fact, in applications calling for such steep rolloffs, many designers find digital filters significantly easier to develop. Prototypes can be easily altered through software changes. Also, software simulations of digital filter designs reflect the exact filter performance, whereas computer simulations of analog filters can only approximate true filter performance, since the parameters of analog filters are sensitive to component values that are initially inexact and can vary substantially.

DIGITAL FILTER BASICS

Common digital filter designs fall into two basic categories—non-recursive (finite impulse-response, FIR) and recursive (infinite impulse-response, IIR). Besides straightforward IIR designs, there is a growing interest in types embodying what is known as a lattice topology. But before examining these digital filter types, let us review some digital filter basics.

Digital filters are not as difficult to understand as you might at first think. A previous *Analog Dialogue* article introduced the subject to our readers (Vol. 17, Number 1, 1983, page 3); the references listed at the end of both that article and this one can provide greater detail.

Although filtering is often required for smoothing signals in the time domain, most designers understand the operation of a filter best in the frequency domain. The spectrum of the input signal is multiplied by the frequency response of the filter to produce an output signal with an altered spectrum. This multiplication in the frequency domain is equivalent to convolution of the waveform and a response function in the time domain. What then is convolution?

To understand the process, first consider a transfer function, H(f), with an ideal magnitude graph in the frequency domain as shown in Figure 1a. The function H(f) responds with unity gain to signals having frequency components from 0 Hz to f_1 Hz, where each frequency component is simply a cosine wave at a particular frequency. For instance, the signal, $\cos(2\pi 3t)$, represents a unity-amplitude frequency component at f = 3 Hz.

Figure 1b illustrates the spectrum of a signal, X(f), whose time value is $\cos(2\pi f_2 t) + \cos(2\pi f_3 t)$. X(f) therefore represents the sum of two equal components at f_2 and f_3. If you want to extract the f_2 component, leaving behind the f_3 component, you could simply pass the X(f) signal through a low pass filter. In fact, H(f) depicts just such a filter, with a cutoff frequency of f_1. Since H(f) equals 1 at f_2 and 0 at f_3, multiplying H(f) by X(f) gives you $1 \times \cos(2\pi f_2 t) + 0 \times \cos(2\pi f_3 t)$, or simply $\cos(2\pi f_2 t)$.

So far, we have been discussing continuous functions of time. However, in digital filters, we are dealing with sampled data, where a function of time consists of a finite number, k, of discrete values, x(n), per second, where k is the sampling rate and n/k is the discrete variable corresponding to time. Thus, a cosine waveform, in discrete time, is expressed as $\cos(2\pi fn/k)$.

Reprinted from Analog Dialogue 17-2 1983

The continuous Fourier transform provides a means for mapping continuous functions of time into the continuous complex frequency domain, and the inverse Fourier transform maps functions of frequency into the time domain. Similarly, the discrete Fourier transformation maps discrete functions of time into the discrete frequency domain, and its inverse transforms discrete functions of frequency into the discrete time domain.

If the function of frequency is the product of two functions—for example, the frequency content of a signal and the transfer function (i.e., the frequency response)—the corresponding time function is the same as the *convolution* of two functions in the time domain—i.e., the signal's time waveform and a time-response function, determined by the transfer function.

Thus, Fourier's theorem, which equates multiplication in the frequency domain to convolution in the time domain, provides a means of calculating the time response directly. As a consequence, the discrete-time convolution:

$$y(n) = [h * x](n) \quad (1)$$

is equal to the sum of the products of the signal and the frequency response, i.e.,

$$y(n) = \sum_{m=1}^{N} h(m) \cdot x(n - m) \quad (2)$$

for all values of n.

Equation 2 represents a series of multiplications and additions, which, if performed in a particular order, will automatically treat the input signal x(n) as if it were put through a low-pass filter. The equation assumes that h(n) is zero for $m < 1$ and for $m > N$, which happens to always be true for FIR filters, where N is the number of samples in h(n).

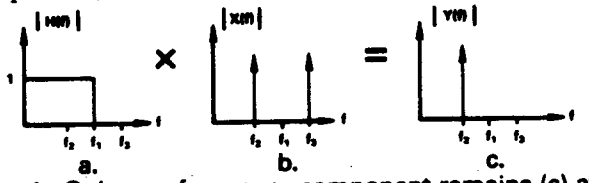

Figure 1. Only one frequency component remains (c) after the filter function (a) multiplies the signal at (b).

To perform the calculation of equation 1, using the Fourier theorem, all you need are the functions h(n) and x(n). These are the inverse discrete Fourier transforms of H(f) and X(f) of Figure 1. The transform of X(f) is a simple cosine wave, $x(n) = \cos(2\pi f_2 n/k) + \cos(2\pi f_3 n/k)$. As discussed in a later section, you can readily calculate the values of x(n) if you know f_2, f_3, and the sample rate, k – the rate at which your analog-to-digital converter is sampling the incoming time-domain signal x(t). You may have a little more difficulty computing the values of h(n), which are called the filter coefficients. But several good computer programs are available to help out, including one from Analog Devices.

To illustrate a practical example of equation 2, consider a 27th order filter, with N = 27. Then, the filter output value y(30), which depends on the 27 preceding values of x, will be:

$$y(30) = h(1)\cdot x(29) + h(2)\cdot x(28) + h(3)\cdot x(27) + \ldots + h(26)\cdot x(4) + h(27)\cdot x(3).$$

The physical meaning of this summation is that the filter's step response is synthesized by summing 27 successively delayed versions of the input step, each multiplied by its own coefficient, in effect building an arbitrary step response. For example, if each h(m) is a gain of 1/27, the filter's response to a step will be a 27-step staircase (approximating an analog ramp), followed by constant output; with any input sequence, it performs a 27-interval running average.

The following sections discuss means for calculating the coefficients.

DIGITAL FILTER TYPES

Figure 2 illustrates FIR and two of the most-prominent IIR digital filter topologies, the former straightforward and the latter in the form of a lattice. FIR, or finite impulse response, filters (Figure 2a) have no feedback terms. Their outputs are a function only of a finite number of previous input values (x(n)), and they are by definition nonrecursive. The filter in the example above is an FIR filter. The IIR filters of 2b and 2c will be seen to have recursive terms, in which a value of the output is affected by previous values of the

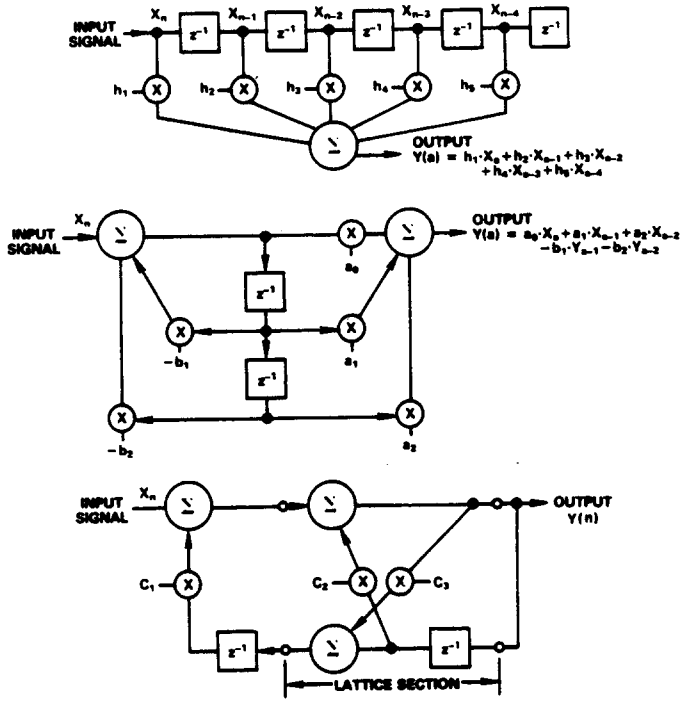

Figure 2. Three common digital filter topologies include FIR (a), IIR (b), and lattice type (c).

output (y(n)), as well as by input values. In comparison to the other types, FIR filters offer:

Stability. FIR filters have no poles in their Z-plane transfer function. Thus, their output is always finite and stable. IIR filters in contrast, require careful design to insure stability. Because FIR designs are based on discrete time delays and have no poles, they can be used to construct filters for which there are no continuous analog equivalents.

Linear Phase Response. You can design FIR filters with linear phase response – the phase delay of the output signal increases linearly with the frequency of the input signal. Linear phase response becomes particularly important in applications such as speech processing, sonar and radar. IIR filters, on the other hand, have nonlinear phase response. Linear phase response is difficult to achieve with continuous analog filters.

Ease of Design. Designers find the FIR filter the easiest of the three forms to understand, design, and implement, especially for indicial response in the time domain.

Low Sensitivity to Coefficient Errors. This permits FIR filters to be implemented with small word sizes – 12-to-16 bits for instance. Typical IIR filters need 16-to-24 bits per coefficient.

Accomodates Adaptive Designs. Adaptive FIR filters are comparatively easy to implement, by changes to the filter coefficients in real time, to adapt the filter's characteristics to external conditions. Adaptive equalization filters in modems, for instance, are programmed to change their characteristics in response to changes in the impedance of the transmission line.

IIR* (infinite impulse response) filter outputs (Figure 2b) combine input values with previous output values, which have been fed back into the circuit. IIR filters are therefore recursive. As in any feedback circuit, to avoid instabilities, IIR filter designs must avoid positive feedback with gains equal to or greater than 1. IIR designs linear phase shift, and they need large coefficient word sizes to keep rounding errors small and insure stability. Nevertheless, IIR filters have major advantages, including:

Highest Efficiency. IIR designs require fewer filter coefficients, thereby minimizing the number of multiplications and maximizing the throughput.

Least Memory Storage. Because the IIR filter has the least number of coefficients, it requires the least amount of read-only memory (ROM). For example, a typical highpass design requires only four coefficients in an IIR implementation, versus 19 for an FIR equivalent.

Lattice-type digital filters promise greater stability than IIR forms, with less hardware than FIR types. The newest form of digital filter, lattice filters presently have rapidly developing design theory. Although earlier lattice designs were highly sensitive to coefficient accuracy, recent designs have shown less sensitivity to filter parameters than the corresponding IIR filter (by 2 to 3 bits!). A big advantage of lattice filters is that the parameters used in each of the steps can be used for efficient encoding methods, as in linear predictive coding (speech).

DESIGNING FIR FILTERS

Specifications and Tradeoffs

Designers specify non-recursive (i.e., FIR) digital filters similarly to analog filters – a maximum amount of ripple in the passband, a maximum amount of attenuation in the stopband, etc. (See the adjacent definitions of digital filter terminology.) You will need to specify the following design parameters:

N, the number of taps in the filter, which equals the number of filter coefficients

f_p, the passband cutoff frequency

f_s, the stopband cutoff frequency,

$K = (\delta_1/\delta_2)$, the ratio of the ripple in the passband to the ripple in the stopband.

Figure 3 illustrates these parameters for lowpass, highpass and bandpass filters. Designers usually define the units of passband ripple in dB as $20 \log_{10} (1 + \delta_1)$, and the units of stopband ripple, also in dB, as $-20 \log_{10} (\delta_2)$. Passband ripple typically ranges from 0.001 to 1 dB, and stopband ripple from -10 dB to -90 dB. Frequencies f_p and f_s are normalized frequencies, which equal the ratio of the actual signal frequency to the sampling frequency. Consider, for example, a filter designed for a sampling frequency of 100 kHz,

*See Johnson, Matt, "Implement Stable IIR Filters Using Minimal Hardware." EDN, April 14, 1983, pp. 153-166.

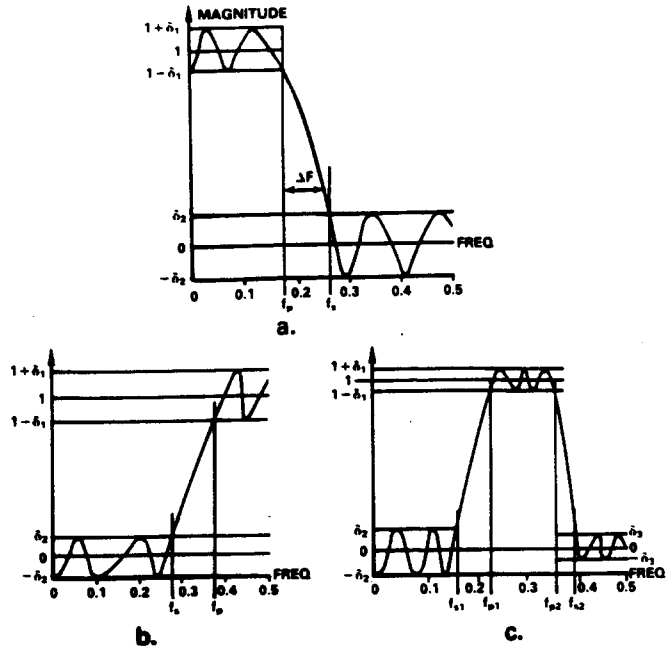

Figure 3. Design parameters defined for lowpass (a), highpass (b), and bandpass (c) filters.

a passband cutoff frequency (f_p) of 10 kHz, and a stopband cutoff frequency (f_s) of 20 kHz. Then,

f_p (normalized) = 10 kHz/100kHz = 0.1

f_s (normalized) = 20 kHz/100kHz = 0.2

Note that the normalized frequency axis extends from 0 to only 0.5, since a design in accordance with the Nyquist sampling theorem requires that a signal be sampled at more than twice its highest frequency in order to eliminate the possibility of aliasing.

As always, specifying these design parameters requires some tradeoffs. With a fixed number of filter taps, steeper rolloffs result in greater ripple. For both steep rolloffs and small ripple, you will have to increase the number of filter taps, and therefore the filter's complexity.

Designing FIR Filters Through Windowing

To design a digital filter, you must first calculate the filter's coefficients, h(m), in order to implement equation 2. The two most common design methods include "windowing" and the Remez Exchange algorithm. For almost 95% of design examples, Remez Exchange results in a significantly more efficient filter. The Remez Exchange algorithm has also been coded in Fortran, and is available from Analog Devices, as noted below.

Windowing methods are useful, however, because of their simplicity, and because they also aid in understanding filtering methods, so they are well worth examining. Keep in mind, though, that FIR designs developed through windowing do not perform as well as those obtained through other methods (see Ref. 7, for instance).

Consider the case of an FIR lowpass filter with stopband attenuation greater than 50 dB, normalized passband cutoff frequency (f_p) of 0.2, and normalized stopband cutoff frequency of 0.3. Figure 4 plots the filter's ideal transfer function, H(f). You can obtain the Fourier series coefficients by solving the inverse Fourier transform by equations 3.

$$h(n) = \int_{-0.5}^{0.5} H(e^{j2\pi f}) \cdot e^{j2\pi fn}\, df$$

$$= \int_{-f_1}^{f_1} e^{j2\pi fn}\, df$$

$$= \int_{-f_1}^{f_1} (\cos 2\pi fn + j \sin 2\pi fn)\, df$$

$$= \frac{\sin(\pi f_1 n)}{\pi n} \qquad (3)$$

Figure 5a shows the resulting set of h(n), which extend to ± infinity. You next multiply the Fourier coefficients by one of several window or weighting functions, as illustrated by Figure 5b and 5c.

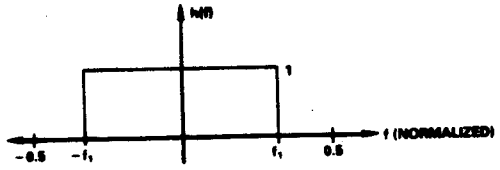

Figure 4. Idealized low-pass filter transfer function

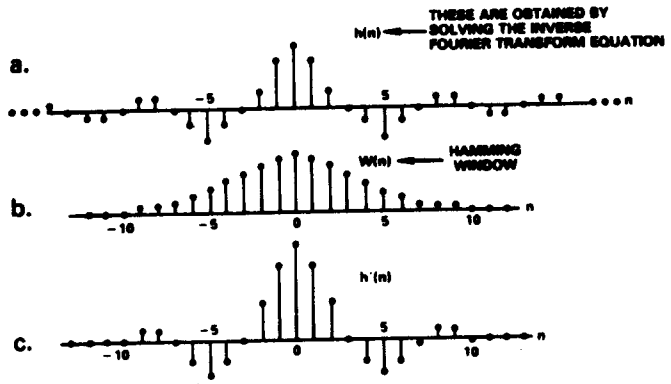

Figure 5. In the window method, a filter's Fourier coefficients (a) multiply a weighting function (b) resulting in (c).

The weighting function is equal to zero above and below some value, ν, which depends on the number of filter taps, N. Multiplying the Fourier coefficients by the weighting function generates a finite impulse response approximation to the desired transfer function, H(f). This guarantees that the Fourier series will converge.

Although several weighting functions will work, Figure 5b plots the widely used Hamming window. Others include the Kaiser, Blackman, and Hanning windows (see Ref. 1). After you choose your window, you can determine the number of coefficients or filter taps, N, from the desired rolloff band, $\Delta f = f_s - f_p$. For the Hamming window, this rolloff bandwidth relates to filter taps, N, by the conservative approximation:

$$\Delta f \approx 4/N \qquad (4)$$

For this design example,

$$\Delta f = f_s - f_p = (0.3 - 0.2) = 0.1.$$

Thus, N = 4/0.1 = 40. This approximation usually yields 2 to 5 more taps than needed, so specify N as 36.

Next, you obtain the filter coefficients, h'(n), of Figure 5c by multiplying each h(n) by the corresponding weighting function w(n) of Figure 5b. Since the coefficients are symmetrical about 0, you need only compute their absolute values (i.e., half the coefficients). The coefficients describe the windowed function, of the form (sin x)/x, which is the Fourier transform of the low pass filter of Figure 4.

Remez Exchange Design

For most FIR applications, the Remez Exchange algorithm offers a more-powerful design technique than the windowing method.* The Remez Exchange algorithm designs an optimal FIR filter as defined by the minimax error criterion (Ref. 7). The minimax criterion specifies a filter that, for a given number of coefficients, *minimizes* the *maximum* ripple in the passband.

In general, for a given set of filter specifications, the Remez Exchange algorithm quickly generates an FIR design with the smallest possible number of filter coefficients, particularly in comparison to the results of the windowing method. Also, passband ripples all have equal amplitude, as do all stopband ripples. You designate the ratio, K, of stopband to passband ripple, as noted above.

The Fortran-coded Remez Exchange algorithm is easy to use. Consider, for instance, the case of an FIR low-pass filter with the following specifications:

Sample rate = 50 kHz
f_p (actual) = 10 kHz, f_p (normalized) = 0.2
f_s (actual) = 14 kHz, f_s (normalized) = 0.28
minimum stopband attenuation = 40 dB
maximum passband ripple = 0.2 dB
ripple ratio K = 1 (equal ripple in pass- and stop-bands)

The Fortran program then prompts the user for inputs using five consecutive lines:

Line 1:

FILT	=	number of filter taps or coefficients. Set this equal to zero if the filter order (number of taps) is not known, as in this design example.
JTYPE	=	type of filter (set to 1 for low-pass, high-pass, or band-pass filters).
NBANDS	=	number of pass-bands plus stop-bands in the filter. For a low-pass filter or a high-pass filter as in this example, NBANDS = 2. A band-pass filter has NBANDS = 3.
JPUNCH	=	(normally set to zero).
LGRID	=	number of frequency points used in the Remez Exchange algorithm. For most applications, such as in this example, LGRID = 16 suffices. For high-performance filters with more than 50 taps, set LGRID to 32.

Line 2:

Line 2 contains the normalized frequencies of the pass-band and stop-band edges. The number of values here equals twice the number of bands. For the case of the low-pass filter specified above, the pass-band ranges from 0.0 to 0.2 in normalized frequency, and the stop-band edge spans 0.28 to 0.5 in normalized frequency. Line 2 therefore carries these four numbers for this design example.

Line 3:

Line 3 specifies the magnitude of the desired transfer function,

V_{out}/V_{in}, in each band. In this example, the low-pass filter has unity gain in the pass-band, and zero gain in the stop-band, so Line 3 contains the numbers 1, 0.

Line 4:

Line 4 specifies the desired relative weights of the two bands. For this example, specify stop-band ripple equal to pass-band ripple, denoted by a 1, 1 on Line 4.

Line 5:

You need Line 5 only if the number of filter taps needed is unknown – as in this example (NFILT = 0). This line specifies the desired pass-band and stop-band ripple in dB. The program then estimates the number of required filter taps NFILT. Assume in this case that passband ripple does not exceed 0.2 dB, and that stopband attenuation is 40.0 dB. Line 5 therefore includes the numbers 0.2 and 40.0.

With these inputs, the Fortran program estimates the filter order (number of taps) by approximating design relationships between the filter parameters (Refs. 5 and 6). The result usually falls within four taps of the correct number needed.

Figure 6 illustrates a typical computer result. The "filter length," determined by approximation as noted above, equals 24 taps. The "impulse response" gives the filter coefficients, and the next few lines of Figure 6 simply repeat the program input values for band 1, the pass-band, and band 2, the stop-band. The "desired value" indicates the desired filter transfer functions in the pass- and stopbands. The "weighting" of the ripples is 1.00 in the passband and 1.00 in the stopband. The "deviation" is the ripple in each band, which equals 0.011 in the passband and 0.011 in the stopband. The "deviation in dB" represents the decibel value of the "deviation" numbers. "Extremal frequencies" denotes frequencies at which maximum passband and stopband ripple occurs.

```
                FINITE IMPULSE RESPONSE (FIR)
                LINEAR PHASE DIGITAL FILTER DESIGN
                REMEZ EXCHANGE ALGORITHM

                BANDPASS FILTER

FILTER LENGTH =  24

FILTER LENGTH DETERMINED BY APPROXIMATION

***** IMPULSE RESPONSE *****
     H(  1) = -0.10748328E-01 = H(  24)
     H(  2) = -0.18704087E-02 = H(  23)
     H(  3) =  0.15714122E-01 = H(  22)
     H(  4) =  0.47213142E-02 = H(  21)
     H(  5) = -0.25240039E-01 = H(  20)
     H(  6) = -0.13135824E-01 = H(  19)
     H(  7) =  0.41533310E-01 = H(  18)
     H(  8) =  0.28864330E-01 = H(  17)
     H(  9) = -0.69702514E-01 = H(  16)
     H( 10) = -0.71428084E-01 = H(  15)
     H( 11) =  0.16081354E+00 = H(  14)
     H( 12) =  0.43481988E+00 = H(  13)

                BAND  1          BAND  2          BAND
LOWER BAND EDGE    0.000000000      0.280000001
UPPER BAND EDGE    0.200000003      0.500000000
DESIRED VALUE      1.000000000      0.000000000
WEIGHTING          1.000000000      1.000000000
DEVIATION          0.011113827      0.011113827
DEVIATION IN DB    0.193075284    -39.082729340

EXTREMAL FREQUENCIES
   0.0000000   0.0416667   0.0807292   0.1197916   0.1582500
   0.1875001   0.2000000   0.2800000   0.2930208   0.3216665
   0.3607289   0.3997812   0.4414577
```

Figure 6. Remez Exchange program output with NFILT initially = 0.

This initial computer run, with N = 24, results in passband ripple of 0.19 dB, and stopband attenuation of 39.08 dB, which do not meet the design specifications. Repeating the computer run, with successively higher values for N, leads to the acceptable results of Figure 7, for N equal to 27.

Hardware Design

Once fully defined, your filter can be readily implemented in hardware. Figure 8 is a functional diagram of a system implementing the 27-tap filter defined above, assuming 16-bit words. The tradeoffs in selecting word size will be discussed later.

```
                FINITE IMPULSE RESPONSE (FIR)
                LINEAR PHASE DIGITAL FILTER DESIGN
                REMEZ EXCHANGE ALGORITHM

                BANDPASS FILTER

FILTER LENGTH =  27

***** IMPULSE RESPONSE *****
     H(  1) =  0.37293068E-02 = H(  27)
     H(  2) = -0.72388127E-02 = H(  26)
     H(  3) = -0.80835225E-02 = H(  25)
     H(  4) =  0.77674030E-02 = H(  24)
     H(  5) =  0.15854538E-01 = H(  23)
     H(  6) = -0.11497792E-01 = H(  22)
     H(  7) = -0.28403830E-01 = H(  21)
     H(  8) =  0.14817088E-01 = H(  20)
     H(  9) =  0.50319806E-01 = H(  19)
     H( 10) = -0.17288791E-01 = H(  18)
     H( 11) = -0.97808425E-01 = H(  17)
     H( 12) =  0.19058505E-01 = H(  16)
     H( 13) =  0.31538205E+00 = H(  15)
     H( 14) =  0.48032677E+00 = H(  14)

                BAND  1          BAND  2          BAND
LOWER BAND EDGE    0.000000000      0.280000001
UPPER BAND EDGE    0.200000003      0.500000000
DESIRED VALUE      1.000000000      0.000000000
WEIGHTING          1.000000000      1.000000000
DEVIATION          0.008834021      0.008834021
DEVIATION IN DB    0.153468403    -41.078831818

EXTREMAL FREQUENCIES
   0.0000000   0.0446429   0.0848214   0.1250000   0.1607143
   0.1875001   0.2000000   0.2800000   0.2911607   0.3179463
   0.3514283   0.3871424   0.4250887   0.4630350   0.5000000
```

Figure 7. Remez Exchange program output, N = 27 taps.

The anti-aliasing filter of Figure 8 minimizes high-frequency signal and noise components reaching the a/d converter. In many cases, anti-aliasing filters require rolloffs no greater than 6-24 dB/octave. The a/d converter samples the incoming analog signal at a rate equal to about three times the highest input frequency. Although the Nyquist criterion specifies a sampling rate of at least two times the highest frequency, conservative design practice dictates a factor of three. The RAM stores the a/d converter's output. With a 27-tap filter, you will need 27 RAM locations, each 16-bits wide.

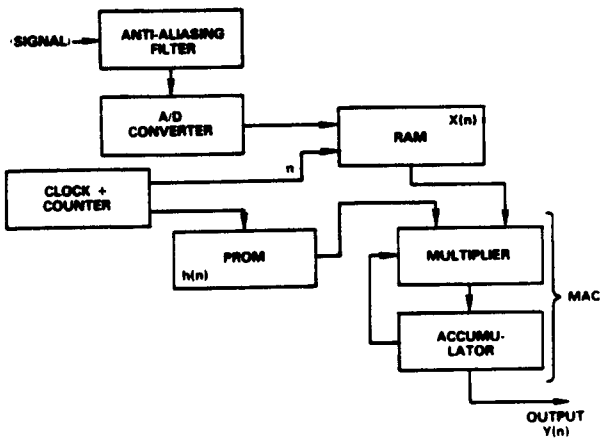

Figure 8. Digital FIR filter system functional block diagram.

A PROM stores the filter coefficients determined earlier. You may need RAM, instead of a PROM, particularly if you wish to implement an adaptive filter. The number of PROM locations equals the number of *different* filter coefficients. Because of symmetry, an FIR filter has N/2 different coefficients for N even (N is the number of taps), and (1 + N)/2 coefficients for N odd. A 27-tap filter therefore requires a PROM with 14 16-bit locations.

The clock and counter step through the RAM and PROM, presenting coefficients and input values to the multiplier. The multiplier/accumulator combination performs the multiplication and addition as specified by equation 2, and thus forms the heart of the digital filter.

Analog Devices offers a variety of multiplier/accumulator ICs which considerably simplify such digital filter implementations. The ADSP-1010*, for instance, multiplies two 16-bit numbers and accumulates the products in a 35-bit accumulator, which includes 3 bits of extended precision to accomodate overflows resulting from the addition of two or more 32-bit products.

Hardware Details

As a first step in implementing a detailed design, convert the filter coefficients to 16-bit fixed-point or block-floating point numbers. In fixed-point arithmetic, for instance, simply multiply the coefficients by 2^{15}.

Next, round off the coefficients to the nearest least-significant bit. Do not simply truncate the coefficients, since truncation destroys the accuracy of the filter coefficients, whereas rounding achieves performance close to the theoretical limits imposed by your word length. Store the rounded 16-bit coefficients in PROM.

You also must determine whether a standard μP can implement the filter, or whether you will need a dedicated high-speed multiplier IC. To determine the required computational speed, multiply the sampling rate by the number of filter coefficients.

In the above example, a sampling rate of 50 kHz and a filter with 27 taps requires (50kHz × 27) = 1.35 million 16-bit multiply-and-accumulate operations per second, or 740 nanoseconds per combined operation. Few microprocessors can handle such requirements; for instance, the 12.5-MHz version of the Motorola 68000 performs a 16-bit multiplication in 5.6 μs. The Analog Devices ADSP-1010 multiplier/accumulator (MAC), however, readily performs a multiply-and-accumulate operation in only 165 nanoseconds, at low cost and with low power consumption.

To ensure proper multiplications, the memory-control circuitry (RAM, PROM, counter) must retrieve the correct combination of words from memory. The stack and pointers of Figure 9 illustrate one method. Pointer 2 directs the storage of each new data point into RAM. For each new sample, the system computes the transformation by incrementing down from pointer 1 and up from pointer 3 as follows:

$$h(4)\cdot x(n-3) + h(3)\cdot x(n-2) + h(2)\cdot x(n-1) + h(1)\cdot x(n) + h(6)\cdot x(n-5) + h(5)\cdot x(n-4).$$

After computing each sample, pointers 2 and 3 increment; the pointers reset when they reach the stack boundary. Figure 9b details the operation.

Next, decide how to handle accumulator overflow. When a filter performs its multiply-and-accumulate operations, the number of bits in the accumulator will certainly exceed the 32-bit resolution of a single 16 × 16-bit multiplication. To handle overflow, first calculate a reasonable upper bound for the amount of overflow your filter could experience. By summing the squares of the filter coefficients, you can estimate a reasonable level of overflow. Compare this number to the maximum the accumulator can handle.

You can handle accumulator overflow in one of several ways. The ADSP-1010 MAC provides three additional bits of accumulator

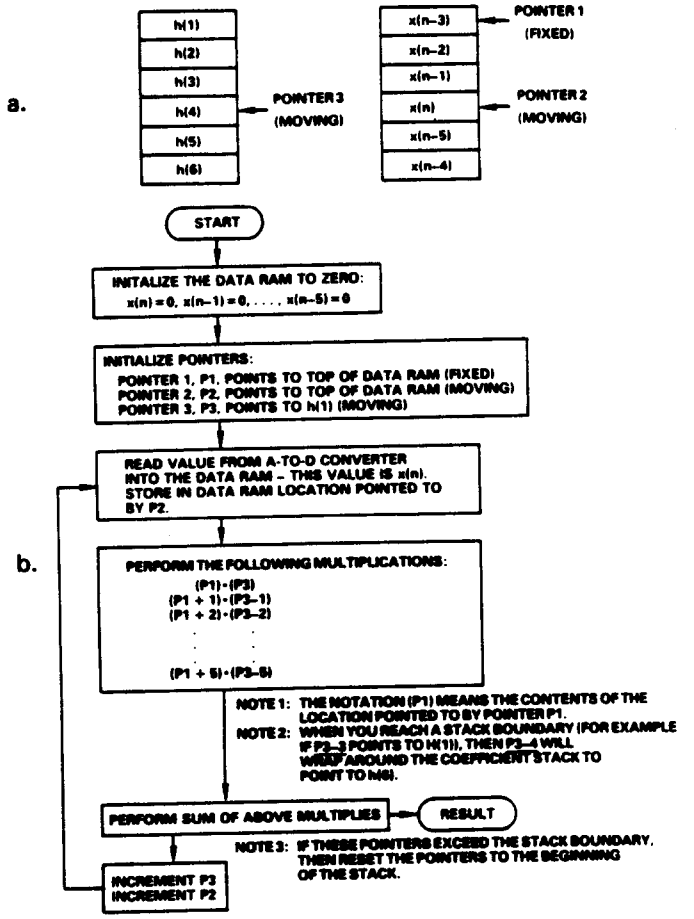

Figure 9. Filter uses pointers (a) to compute convolution as outlined in (b).

precision in addition to the 32 bits needed to handle a single 16 × 16-bit multiplication. This suffices for most applications.

Alternatively, you can scale down the coefficients, from 1 to 5 bits, at the sacrifice of some accuracy. To scale them down, divide them by 2 and apply the overflow test described above. Continue the process until the scaled coefficients pass the overflow test.

Finally, for some applications, you may not want to accomodate the full dynamic range of the input signal. Therefore, just let the accumulator saturate at its maximum value.

Occasionally, required multiply/accumulate speeds exceed the capabilities of even the fastest MAC ICs. In that case, you can combine two or more processors in parallel to increase the throughput. The circuit of Figure 10 combines two ADSP-1010 MACs operating in parallel, thus cutting the multiply/accumulate time per computed point to 75 nanoseconds, which is one-half the normal 150 nanoseconds for a single such component.

Avoid Rounding and Roundoff Errors

Most digital filter hardware errors result from two sources – rounding and roundoff. *Rounding* errors result from the rounding of filter coefficients, such as those generated in Figures 6 and 7,

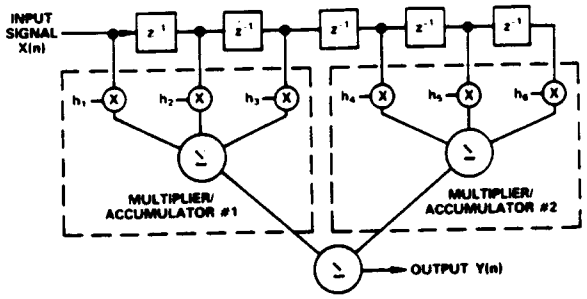

Figure 10. Paralleling two multiplier/accumulators doubles throughput.

by a high precision mainframe computer, to the 16-bits of typical digital filter hardware implementations. As noted earlier, rounding produces less error than truncating, but error nonetheless.

Roundoff errors result from consecutive finite precision multiply-and-accumulates. Roundoff errors are more significant than rounding errors, particularly in high-order filters.

Estimating the required word size to avoid such errors can prove tricky. Generally, if your design calls for more than 67 dB of stop-band attenuation or less than 0.05 dB of passband ripple, 16-bit words may lead to excessive errors. Such cases may require 24, and sometimes even 32-bit, word lengths. Software simulation, discussed in a following section, can help you determine word-length requirements before you commit your design to hardware.

To illustrate the significance of these errors, Figure 11 compares simulated performance of 16-bit fixed-point and 32-bit floating-point 27-tap low pass filters. Although the errors appear slight in this case, a similar comparison in Figure 12 for a 90-tap filter shows dramatic differences. For more than 80 dB of stop-band attenuation with 90 taps, more than 16 bits of precision are needed.

Software Simulations

The flow chart of Figure 13 shows a typical software program for simulating the performance of your digital filter with a high-resolution computer. You can obtain from Analog Devices a Fortran-coded version of this program for simulating 16-bit FIR designs, employing the ADSP-1010 16 × 16-bit multiplier/accumulator. It is available from the DSP Marketing Group, under the name, "FIR 16-bit simulation program."

The simulation repeats the steps in the hardware design process. It begins by obtaining the filter coefficients h(n) from the Remez-Exchange computer program, checks for overflow and scales the coefficients, and obtains the 16-bit fixed-point or floating-point coefficients, normally stored in PROM.

The program next simulates a digitized input signal array, x(n), which corresponds to the output of the A/D converter, normally stored in RAM. The number of values in the array equals the number of filter taps. Normally, you should begin the simulation with a cosine wave of frequency 0 Hz, and work you way up to higher frequencies.

The arithmetic operations of the multiplier/accumulator combination are readily simulated. The simulation program restricts the computer's word size to correspond to the limited precision (16 bits) of your filter's hardware implementation. In Fortran, for example, the INTEGER*2 or INTEGER*4 variable type declarations handle this for you.

The simulation program also includes an accumulator overflow check to verify the effectiveness of the initial coefficient scaling operation. If the computer flags an accumulator overflow, you'll have

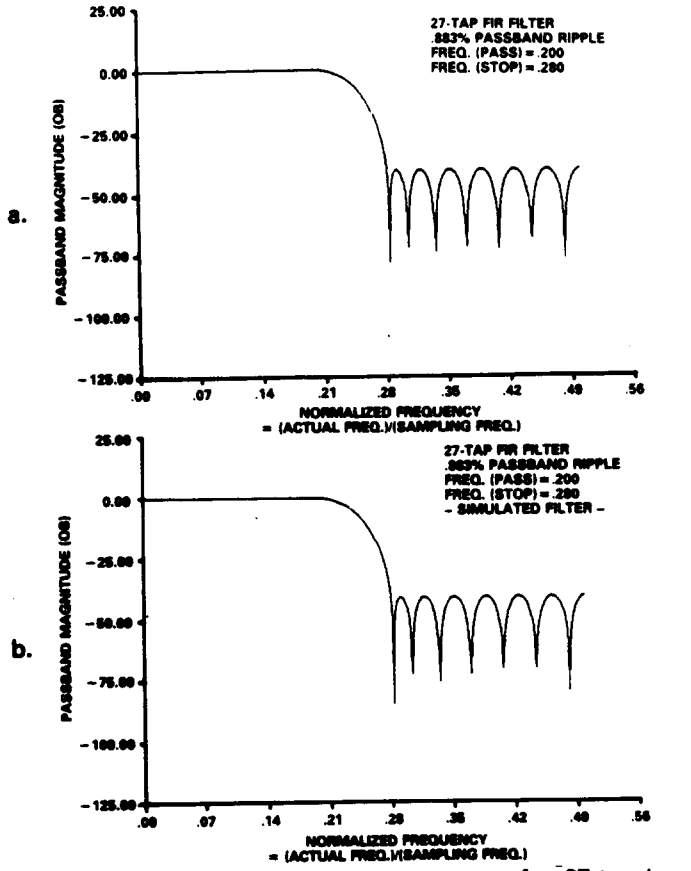

Figure 11. Computer-simulated response of a 27-tap low pass FIR filter using 32-bit arithmetic (a), and 16-bit arithmetic (b).

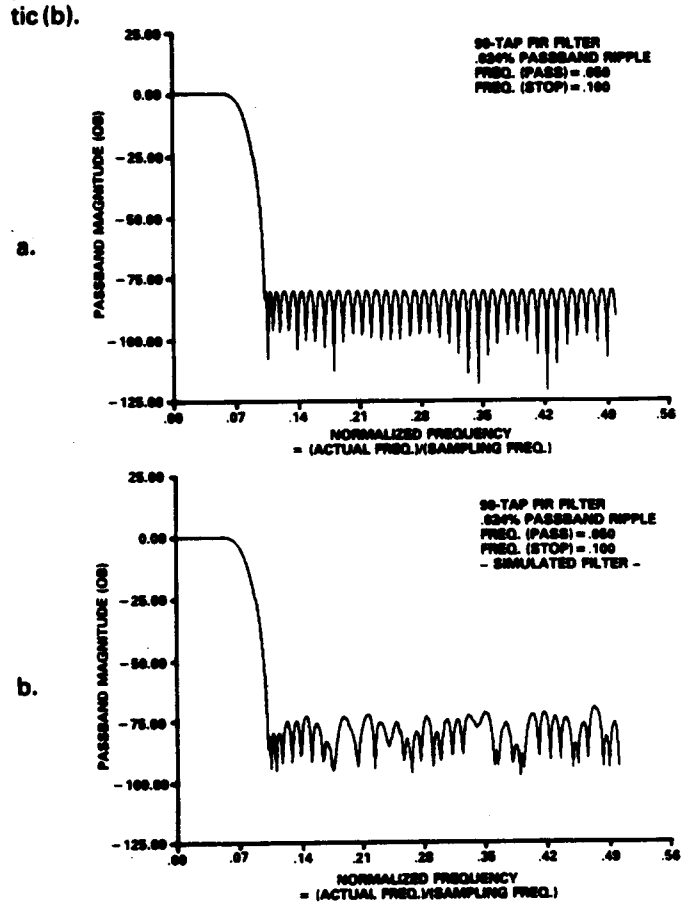

Figure 12. Computer simulated response of a 90-tap low pass FIR filter using 32-bit arithmetic (a), and 16-bit arithmetic (b).

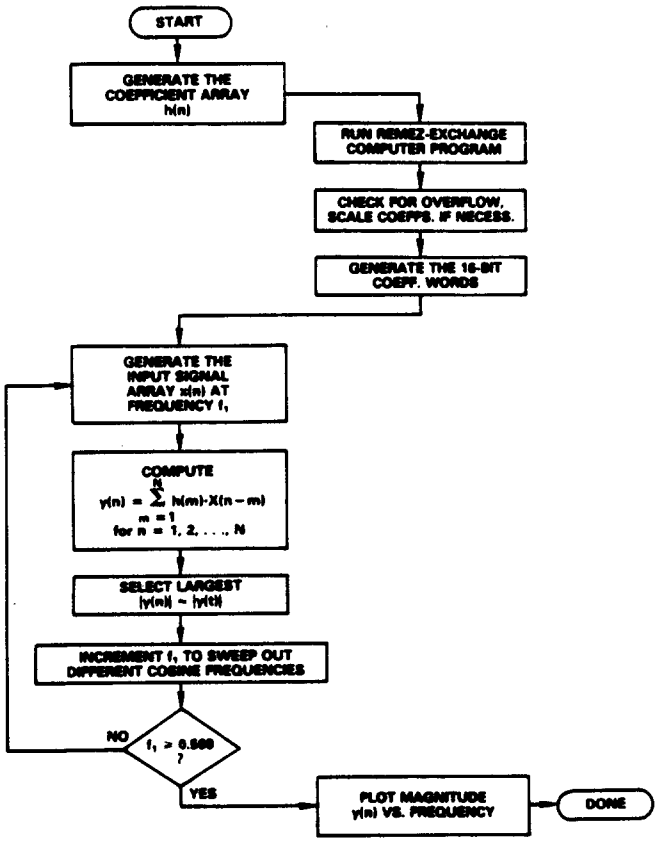

Figure 13. FIR filter simulation program flowchart.

to scale down the coefficients again and re-run the simulation.

The program next computes the filter output values, y(n), by setting up a loop to calculate the transformation of equation 2. For each cosine input, the program computes N values of y(n), where N is the number of filter taps.

The program next finds the magnitude of the filter's output (and therefore the magnitude of the filter's transfer function at that frequency), by choosing the largest absolute value from the y(n) array. This usually comes very close to the actual magnitude of y(n). But if you need better accuracy, you can pass the N points of the y(n) signal through a curve-fitting algorithm which generates a continuous-time signal, y(t).

Once the program computes the output for a cosine wave of 0 Hz, it can calculate outputs for a range of frequencies. You usually want it to sweep from 0 Hz to just below the Nyquist frequency of 0.5 (normalized), in normalized-frequency increments of 0.001. The resulting plot simulates your filter's transfer function. If the plot meets your expectations, you can proceed with the construction of the hardware. ◘

FOR FURTHER READING

Analog Dialogue, Volume 17, Number 1 (1983) carried a review article summarizing the uses of multiplier/accumulator ICs in a variety of DSP applications. Analog Devices has also collected a recently published series of 5 articles comprising cookbook designs applying DSP to various common filtering, and control applications. If you would like reprints of this set of articles, use the reply card; ask for "EDN series."

Ted Dintersmith, and Paul Toldalagi, "Apply Modern Control Theory to Optimize Digital Systems," *EDN*, April 28, 1983, pp. 165-179.

Matt Johnson, "Implement Stable IIR Filters Using Minimal Hardware," *EDN*, April 14, 1983, pp. 153-166.

John Oxaal, "Temporal Averaging Techniques Reduce Image Noise," *EDN*, March 17, 1983, pp. 211-215.

John Oxaal, "DSP Hardware Improves Multiband Filters," *EDN*, March 31, 1983, pp. 193-197.

Bill Windsor, and Paul Toldalagi, "Simplify FIR-Filter Design With a Cookbook Approach," *EDN*, March 3, 1983, pp. 119-128.

REFERENCES

The following publications should prove useful to the reader seeking even more information on digital signal processing. These publications are *not* available from Analog Devices.

1. Frederick J. Harris, "On the Use of Windows for Harmonic Analysis With the Discrete Fourier Transform," *Proceedings of the IEEE*, Vol. 66, No. 1, January, 1978.

2. J. H. McClellan, T. W. Parks, and L. R. Rabiner, "A Computer Program for Designing Optimum FIR Linear Phase Digital Filters," *IEEE Transactions on Audio and Electroacoustics*, Vol. AU-21, No. 6, December, 1973.

3. A. V. Oppenheim, and R. W. Schafer, *Digital Signal Processing*, (Englewood Cliffs, New Jersey: Prentice-Hall, 1975), chapter 9.

4. A. Peled, and B. Liu, *Digital Signal Processing*, (New York, New York: John Wiley and Sons, Inc., 1976), chapter 2.

5. L. R. Rabiner, "Practical Design Rules for Optimum Finite Impulse Response Low-Pass Digital Filters," *The Bell System Technical Journal*, Vol. 52, No. 6, July-August, 1973.

6. L. R. Rabiner, "Approximate Design Relationships for Low-Pass FIR Digital Filters," *IEEE Transactions on Audio and Electroacoustics*, Vol. AU-21, No. 5, October, 1973.

7. L. R. Rabiner, and B. Gold, *Theory and Application of Digital Signal Processing*, (Englewood Cliffs, New Jersey: Prentice-Hall, 1975), chapter 3.

FILTER TERMINOLOGY

Attenuation – A decrease in output signal magnitude relative to input signal magnitude.

Cutoff frequency – The frequency at which the filter's response drops below the specified pass-band ripple ($1-\delta_1$).

Pass-band – The filter frequency range through which signals pass without more than a specified amount of attenuation.

Stop-band – The filter frequency range through which signals experience a specified degree of attenuation.

Stop-band attenuation – The minimum amount of attenuation in the stop-band.

Pass-band ripple – The maximum deviation from the desired output magnitude in the pass-band.

Sampling rate – The rate at which the system samples the input signal.

Filter coefficients – Numbers representing the inverse Fourier transform of the filter's transfer function. Coefficients define the filter's characteristics and form the basis of digital filter implementations.

Taps – Taps equal the number of sampled input values processed by the filter for each output point. Taps also equals the number of filter coefficients, and can represent a measure of the filter delay.