SECTION VII

DIGITAL SIGNAL PROCESSING TECHNIQUES

DIGITAL SIGNAL PROCESSING TECHNIQUES

■ DIGITAL FILTERING

Finite Impulse Response (FIR)Filters,

The Duality of the Time and Frequency Domains

FIR Filter Implementation in DSP Hardware Using Circular Buffering

FIR Filter Design Techniques

Filter Design Using CAD Techniques

Design Example for an FIR Digital Audio Filter Using CAD Program

Insuring Linear Phase in FIR Filters

Decimation Using FIR Filters

Infinite Impulse Response (IIR) Digital Filters

Summary: FIR Versus IIR Filters

■ FAST FOURIER TRANSFORMS

FFT Hardware Implementation

FFT Design Considerations

Spectral Leakage and Windowing

Data Scaling and Block Floating Point

SECTION VII

DIGITAL SIGNAL PROCESSING TECHNIQUES

DIGITAL FILTERING

Real-time digital filtering is one of the most powerful tools of DSP. Apart from the obvious advantages of virtually eliminating errors in the filter associated with passive component fluctuations over time and temperature, op amp drift (active filters), etc., digital filters are capable of performance specifications that would, at best, be extremely difficult, if not impossible, to achieve with an analog implementation. In addition, the characteristics of a digital filter can be easily changed under software control. Therefore, they are widely used in adaptive-filtering applications such as modems, digital audio, digital mobile radio, and speech processing.

The actual procedure for designing digital filters has the same fundamental elements as that for analog filters. First, the desired filter responses are characterized and the filter parameters are then calculated. Characteristics such as transfer function and phase response are used in the same way. The key difference between analog and digital filters is that instead of calculating resistor, capacitor, and inductor values for an analog filter, coefficient values are calculated for a digital filter. So for the digital filter, numbers replace the physical resistor and capacitor components of the analog filter. These numbers reside in a memory as filter coefficients and are used along with data values from the ADC in performing the filtering calculations.

The digital filter, because it is a discrete function, works with digitized data as opposed to a continuous waveform, and a data point is acquired each sampling period. Because of this discrete nature, we can reference data samples by numbers such as sample 1, sample 2, sample 3, etc. Figure 7.1, illustrating the basic filtering function, shows a low frequency signal containing higher frequency noise which must be filtered out. This waveform must be digitized with an ADC to produce samples x(n). These data values are fed to the digital filter, which in this case is a lowpass filter. The output data samples, y(n), are used to reconstruct an analog waveform using a DAC.

Digital filters, however, are not the answer to all signal processing filtering requirements. In order to maintain real-time operation, the DSP processor must be able to execute all the steps in the filter routine within one sampling clock period, $1/f_s$. This currently limits their use to primarily voice and audio bandwidth applications. However, it is possible to sacrifice software control and flexibility, and design special hardware digital filters which will operate at video-speed sampling rates. In other cases, the speed limitations can be overcome by first storing the high speed ADC data in a buffer memory. The buffer memory is then read at a rate which is compatible with the speed of the DSP-based digital filter. In this manner, pseudo real-time operation can be maintained as in a radar system, where signal processing is typically done on bursts of data collected after each transmitted pulse. Even in highly oversampled sampled data systems, a simple analog antialiasing filter is usually required ahead of the ADC and after the DAC. Finally, as signal frequencies increase sufficiently, they surpass the capabilities of available ADCs, and digital filtering then becomes impossible, since we no longer have

a sampled data system because we have no ADC. Active analog filtering is not even possible at extremely high frequencies because of op amp bandwidth and distortion limitations, and filtering requirements must then be met using purely passive components. The primary focus of the following discussions will be on filters which can run in realtime under DSP program control.

DIGITAL FILTERING

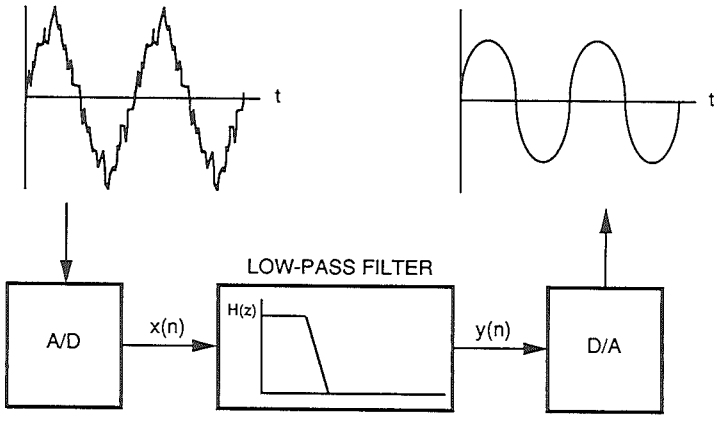


Figure 7.1

DIGITAL FILTERING ADVANTAGES

- High Accuracy
- High Performance
- Linear Phase, Constant Group Delay (FIR Filters)
- No Drift Due to Component Variations
- Flexibility, Adaptive Filtering Possible
- Easy to Simulate and Design

Figure 7.2

DIGITAL FILTER LIMITATIONS

- Computation Must be Completed in Sampling Period
- Limited to Voice and Audio Bandwidth Signals if Real-Time Operation is to be Maintained
- Hardwired Digital Filters Required for Video Frequencies
- Analog Filters Still Needed: Antialiasing and High Frequencies
- Lack of High Speed ADCs for Sampling

Figure 7.3

FINITE IMPULSE RESPONSE (FIR) DIGITAL FILTERS

The simplest form of a digital filter is the finite impulse response filter (FIR), and the most elementary form of an FIR filter is a *moving average* filter as shown in Figure 7.4, where we show a 7-day moving average of a dieter's weight plotted along with the daily weights. After 7 days worth of data samples are obtained, the first point on the moving average is computed by adding the 7 data samples together and dividing by 7. Another way to view the process is to *weight* each data sample by a factor of 1/7 and perform a summation. To obtain the second point on the moving average, the first weighted data sample is subtracted from the summation, and the 8th weighted data sample is added to the summation. This process continues, and can be viewed as a very crude lowpass filtering of the daily readings. The digital implementation of the process is shown in Figure 7.5 which shows the various multiplications, delays, and the summation. The Finite Impulse Response (FIR) filter gets its name because the impulse response is of finite duration; i.e., after seven zero-valued input samples, the filter output goes to zero. When processing an actual electrical signal, a moving average might look like Figure 7.6. It is useful from a mathematical standpoint to view the moving average filter as a *convolution* of the filter impulse response h(t) with the sampled data points x(t) to obtain the output y(t) as shown in Figure 7.7. For a linear convolution, the operation involves multiplying x(t) by a reversed an linearly shifted version of h(t), and then summing the values in the product.

The sin(x)/x frequency response of the moving average filter is shown in Figure 7.8 for various numbers of taps, N. (Note: in this section N refers to the number of sample points and not the number of bits of resolution of an ADC or DAC!). Note that increasing the number of taps sharpens the rolloff characteristic of the moving average filter but does nothing to improve the undesirable sidelobes.

It is possible to dramatically improve the performance of the simple FIR moving average filter by properly selecting the individual weights or coefficients rather than giving them equal weight. The sharpness of the rolloff can be improved by adding more stages (taps), and the stopband attenuation characteristics can be improved by properly selecting the filter coefficients. The essence of FIR filter design is the appropriate selection of the filter coefficients and the number of taps to realize the desired transfer function H(f). Various algorithms are available to translate the frequency response H(f) into a set of FIR coefficients. Most of this software

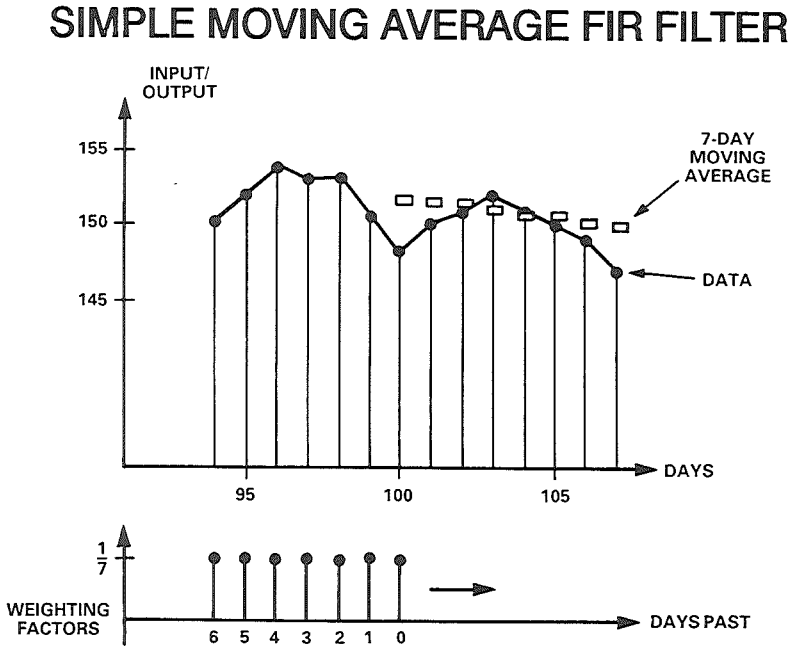


Figure 7.4

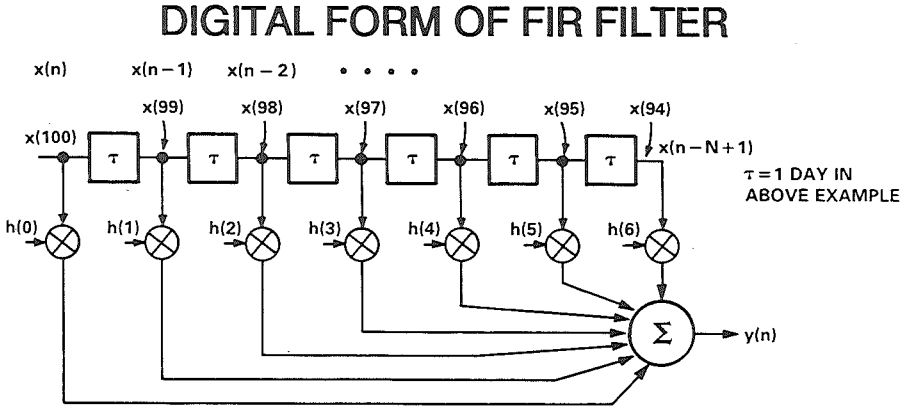


Figure 7.5

MOVING AVERAGE FIR FILTER APPLIED TO ANALOG SIGNAL

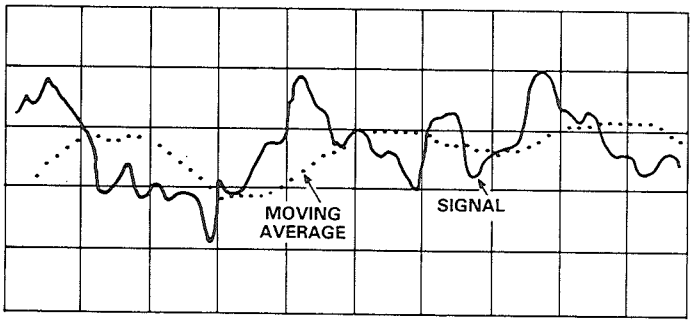


Figure 7.6

MOVING AVERAGE COEFFICIENTS CONVOLVED WITH SAMPLED WAVEFORM

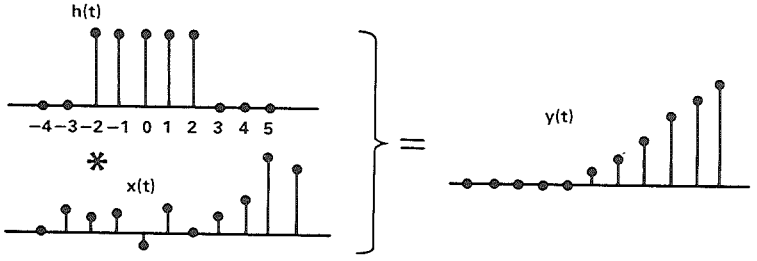


Figure 7.7

FREQUENCY RESPONSE OF MOVING AVERAGE FILTER FOR VARIOUS NUMBER OF TAPS

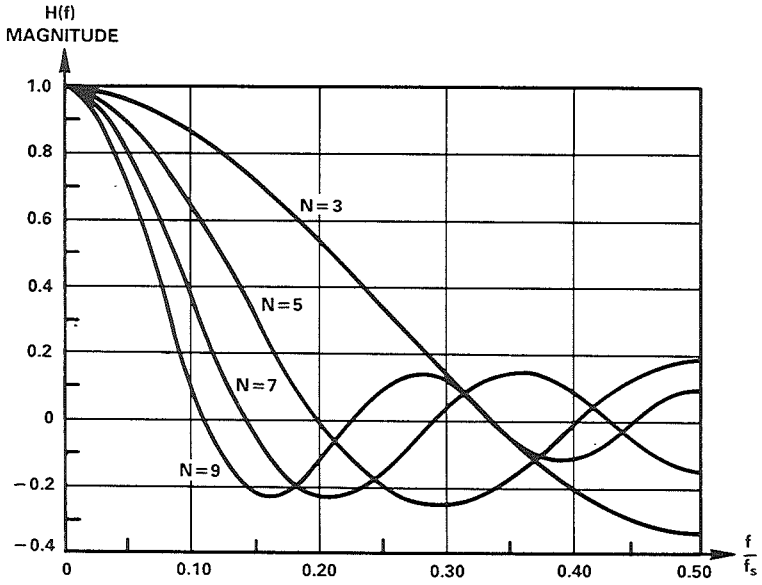


Figure 7.8

is commercially available and can be run on PCs. *The key theorem of FIR filter design is that the coefficients h(n) of the FIR filter are simply the quantized values of the impulse response of the frequency transfer function H(f).* Conversely, the impulse response is the Fourier Transform of H(f).

FACTORS DETERMINING FIR FILTER TRANSFER FUNCTION H(f)

- Number of Taps
- Proper Selection of Weighted Filter Coefficients

Figure 7.9

THE DUALITY OF THE TIME AND FREQUENCY DOMAINS

It is useful to digress for a moment and examine the relationship between the time domain and the frequency domain to better understand the principles behind digital filters such as the FIR filter. In a sampled data system, a convolution operation can be carried out by performing a series of multiplications and accumulations. The convolution operation in the time or frequency domain is equivalent to point by point multiplication in the opposite domain. For example, convolution in the time domain is equivalent to

multiplication in the frequency domain. This is shown graphically in Figure 7.10. It can be seen that filtering in the frequency domain can be accomplished by multiplying all frequency components in the passband by a 1 and all frequencies in the stopband by 0. Conversely, convolution in the frequency domain is equivalent to point by point multiplication in the time domain.

The transfer function in the frequency domain (either a 1 or a 0) can be translated to the time domain by the Fourier transform. This transformation produces an impulse response in the time domain. Since the multiplication in the frequency domain (signal spectrum times the transfer function) is equivalent to convolution in the time domain (signal convolved with impulse response), the signal can be filtered by convolving it with the impulse response. The FIR filter is exactly this process. Since it is a sampled data system, the signal and the impulse response are quantized in time and amplitude yielding discrete samples. The discrete samples comprising the impulse response are the FIR filter coefficients.

The mathematics involved in filter design (analog or digital) most always make use of transforms. In continuous-time systems, the Laplace transform can be considered to be a generalization of the Fourier Transform. In a similar manner, it is possible to generalize the Fourier transform for discrete-time sampled data systems, resulting in what is commonly referred to as the z-transform. Details describing the use of the z-transform in digital filter design are given in References 1, 2, and 3.

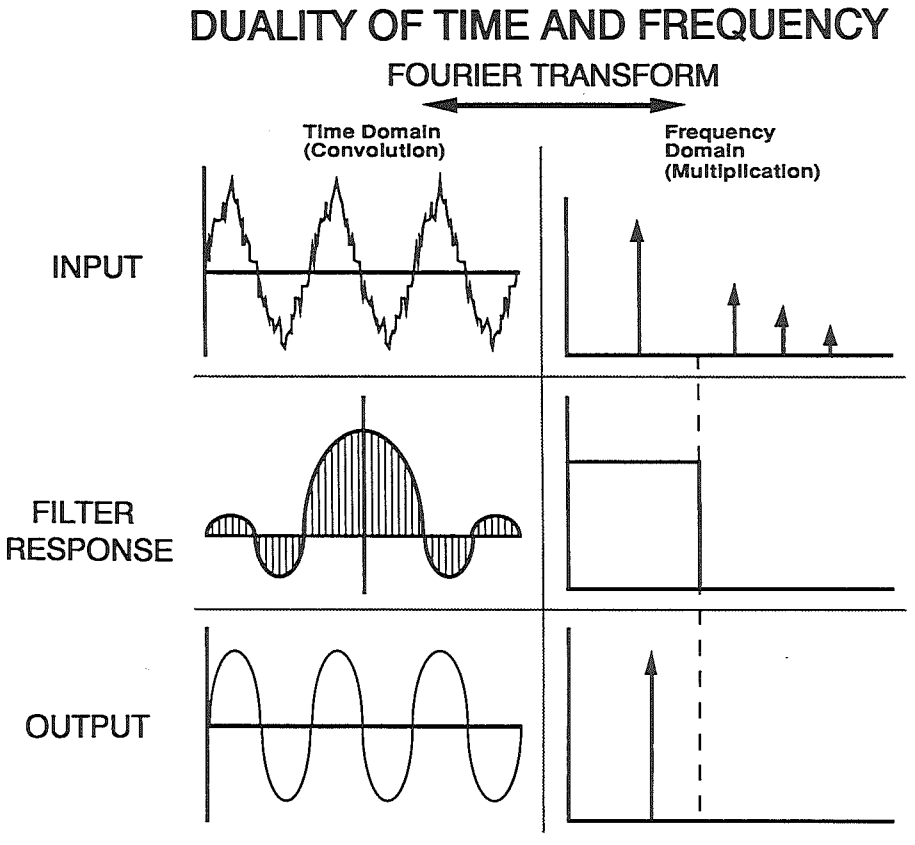


Figure 7.10

FIR FILTER IMPLEMENTATION IN DSP HARDWARE USING CIRCULAR BUFFERING

As has been discussed, an FIR filter (shown in Figure 7.11 must perform the following convolution equation:

$$y(n) = h(n)*x(n) = \sum_{i=0}^{N-1} h(i)x(n-i) \quad ,$$

where h(i) is the filter coefficient array and x(n-i) is the input data array to the filter. The number N, in the equation, represents the number of taps of the filter and relates to the filter performance as has been discussed above.

In the series of FIR filter equations, the N coefficient locations are always accessed sequentially from h(0) to h(N-1). The associated data points circulate through the memory; new samples are added replacing the oldest each time a filter output is computed. A fixed boundary RAM can be used to achieve this circulating buffer effect as shown in Figure 7.12 for a 4 tap FIR filter. The oldest data sample is replaced by the newest after each convolution. A "time history" of the four most recent data samples is kept in RAM.

This delay line can be implemented in fixed boundary RAM in a DSP chip if new data values are written into memory, overwriting the oldest value. To facilitate memory addressing, old data values are read from memory starting with the value one location after the value that was just written. For example, x(4) is written into memory location 0, and data values are then read from locations 1,2,3,and 0. This example can be expanded to accommodate any number of taps. By addressing data memory locations in this manner, the address generator need only supply sequential addresses regardless of whether the operation is a memory read or write. This data memory buffer is called *circular* because when the last location is reached, the memory pointer must be reset to the beginning of the buffer.

The coefficients are fetched simultaneously with the data. Due to the addressing scheme chosen, the oldest data sample is fetched first. Therefore, the last coefficient must be fetched first. The coefficients can be stored backwards in memory: h(N-1) is the first location, and h(0) is the last, with the address generator providing incremental addresses. Alternatively, coefficients can be stored in a normal manner with the accessing of coefficients starting at the end of the buffer, and the address generator being decremented. In the example shown in Figure 7.12, the coefficients are stored in a reverse manner.

FIR FILTER DESIGN TECHNIQUES

FIR filter design calls for specifying a finite set of N coefficients, h(n), to approximate an idealized filter form. *The filter coefficients, h(n), in the time domain correspond to the impulse response of the filter transfer function H(f).*

DIRECT FORM FIR FILTER

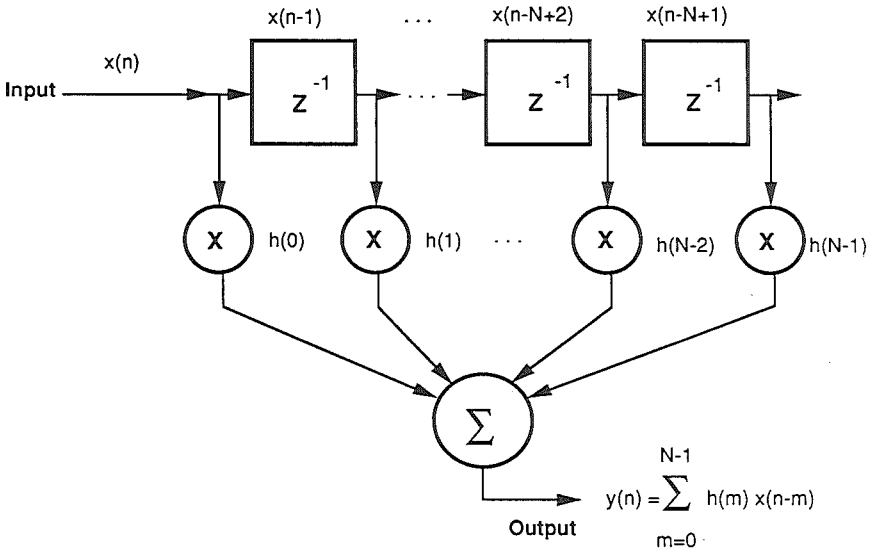
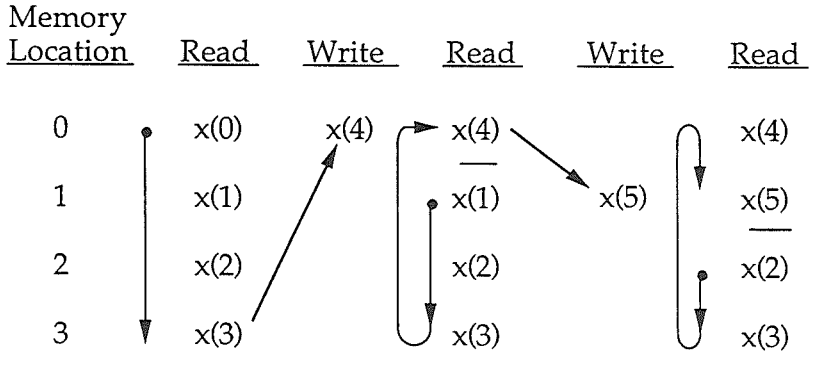


Figure 7.11

DATA MEMORY ADDRESSING FOR 4 TAP FIR FILTER

$$y(n) = h(n) * x(n) = \sum_{i=0}^{N-1} h(i)\, x(n-i)$$



$y(3) = h(0)x(3) + h(1)x(2) + h(2)x(1) + h(3)x(0)$

$y(4) = h(0)x(4) + h(1)x(3) + h(2)x(2) + h(3)x(1)$

$y(5) = h(0)x(5) + h(1)x(4) + h(2)x(3) + h(3)x(2)$

Figure 7.12

KEY FIR FILTER DESIGN THEOREM

- **The Coefficients h(n) of an FIR Filter are Simply the Quantized Values of the Impulse Response of the Frequency Transfer Function H(f)**
- **The Impulse Response is Calculated by Taking the Fourier Transform of H(f)**

Figure 7.13

In Figure 7.14, 2nd, 4th, and 6th-order ideal Chebyshev lowpass filter transfer functions, optimized for 1dB in-band ripple, are compared with a 91-tap (i.e., 91 coefficients and 91 sequential circular buffer memory locations) digital FIR filter optimized for 0.002dB passband ripple. There is no practical analog equivalent; this is higher order than is realistic with analog hardware (greater than 70 poles using rule-or-thumb approximation). Since the response is flatter within the passband, the signal is reproduced more faithfully, and phase distortion in the passband is negligible, since all frequencies are delayed equally by the filter. This is another important characteristic of FIR filters (linear phase response and constant group delay) which makes them extremely attractive to digital audio applications.

If the 91-tap FIR filter shown in Figure 7.14 is implemented in the ADSP-2101 microcomputer, each tap requires one processor cycle (80ns). The total processing time is therefore 7.3μs. This implies that sampling rates of up to about 136kHz can be achieved and still maintain real-time operation.

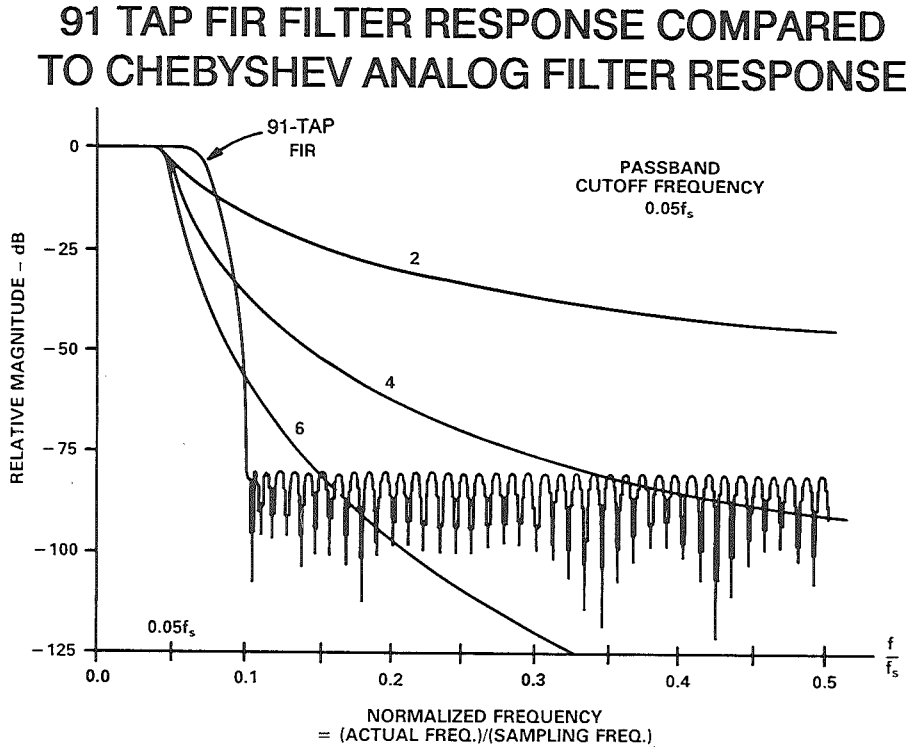


Figure 7.14

91 TAP FIR FILTER PERFORMANCE CHARACTERISTICS

- 0.002dB Passband Ripple
- Linear Phase
- 80dB Stopband Attenuation
- 136kHz Sampling Rate Possible with ADSP-2101 Processor (80ns Cycle Time per Filter Tap)
- No Analog Equivalent! (70 poles Required!)

Figure 7.15

FIR FILTER DESIGN USING CAD TECHNIQUES

In actual practice, the concepts presented in the above discussions have been implemented in easy to use CAD programs which can be run on most PCs. It is only necessary to specify the desired FIR filter characteristics (sampling frequency, passband frequency, stopband frequency, passband ripple, and stopband attenuation) as shown in Figure 7.16. The CAD program calculates the number of filter taps required (N), the impulse response, and the filter coefficients.

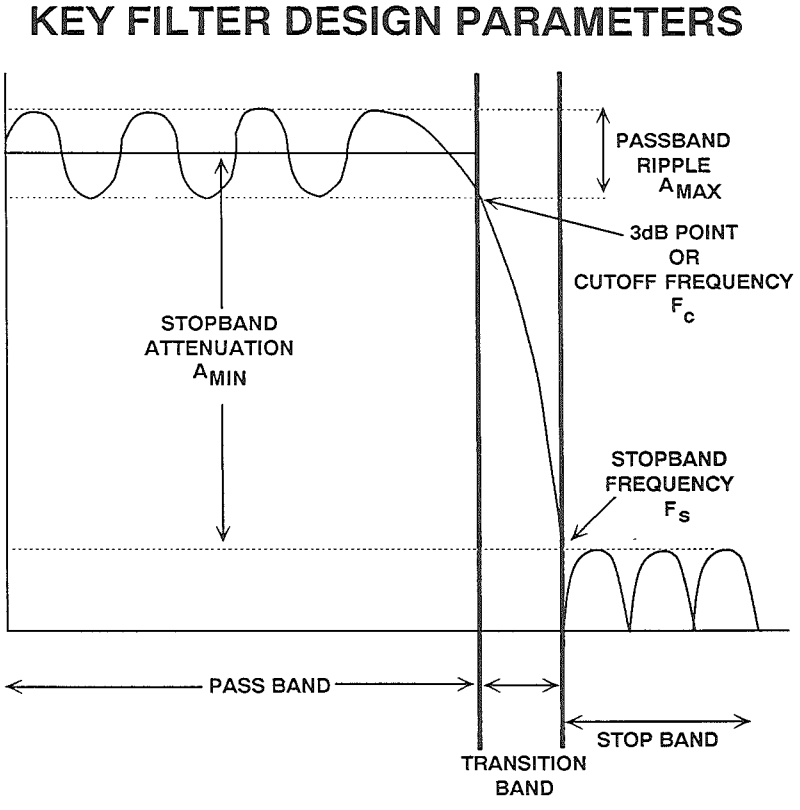


Figure 7.16

FIR FILTER DESIGN CAD PROGRAM INPUTS

- Passband
- Passband Ripple
- Stopband
- Stopband Attenuation
- Wordlength, i.e., 16 Bit Fixed-Point

Figure 7.17

A plot of the frequency response, H(f), along with the impulse response and the step-function response is also available as an output. If the response characteristics are satisfactory, the filter coefficients can then be downloaded into the DSP processor. The CAD program can also simulate the effects of finite word-length (i.e., performing calculations in 16 bit fixed point arithmetic) on the transfer function.

FIR FILTER DESIGN CAD PROGRAM OUTPUTS

- Frequency Response Plot Showing Effects of Finite Wordlength Arithmetic
- Impulse Response Plot
- Step Function Response
- Number of Taps Required
- Filter Coefficients

Figure 7.18

Other algorithms have been developed for CAD filter designs which optimize the filter performance for various characteristics. An example is the Parks and McClellan program (see Reference 1) which minimizes the maximum errors between the desired characteristic and the actual characteristic by using the Remez exchange algorithm from approximation theory.

DESIGN EXAMPLE FOR AN FIR DIGITAL AUDIO FILTER USING CAD PROGRAM

For this example, we will design an audio lowpass filter that is designed to operate at a sampling rate of 44.1kHz (standard for CD players). The program is available from Momentum Data Systems, Incorporated (Reference 5). The program is menu-driven and IBM PC compatible. The filter will be implemented as a Direct Form FIR as shown in Figure 7.19.

First, we select the type of filter to be designed from among the Main Menu shown in Figure 7.20. We choose the Equiripple FIR Design (Parks-McClellan)

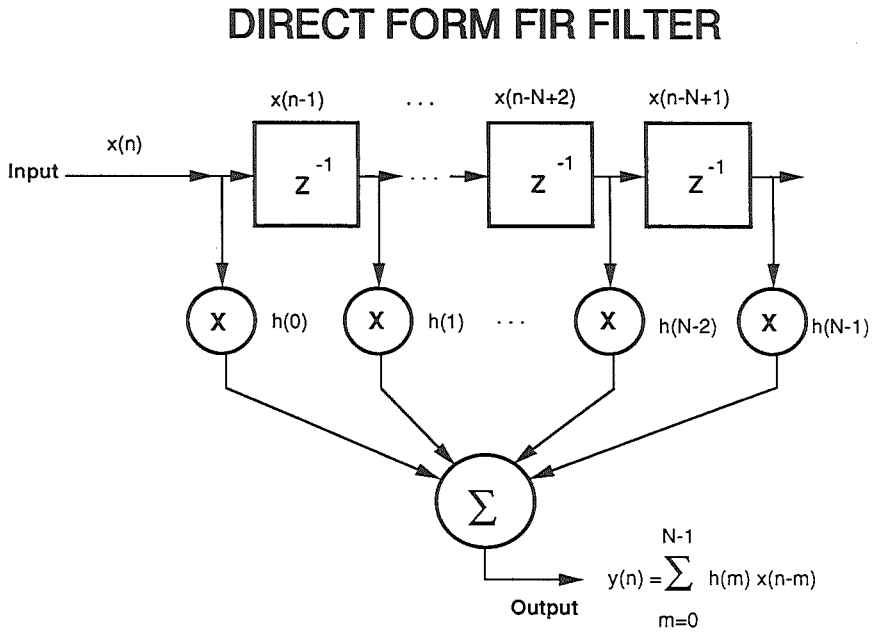


Figure 7.19

FILTER DESIGN AND ANALYSIS SYSTEM MAIN MENU (Screen 1)

- IIR Filter Design
- FIR Filter Design With Windows
- *Equiripple FIR Design (Parks-McClellan)*
- Read Filter Specification File
- System Analysis (Z Domain Input)
- System Analysis (s Domain Input)
- Read System Analysis Input File
- Set System Defaults
- Exit to DOS

Figure 7.20

FINITE IMPULSE RESPONSE FILTER DESIGN MENU (Screen 2)

Filter Type:	1 - *Lowpass*	4 - Bandstop
	2 - Highpass	5 - Differentiator
	3 - Bandpass	6 - Multiband
Frequency Mode:	H - *Hertz*	
	R - Radians/Second	
Gain Specification Mode:	1 - *Maximum Gain 1.0*	
	2 - Nominal Gain 1.0	
Filter Compensation:	Enter X to Select	

Figure 7.21

The second screen then appears as shown in Figure 7.21. This screen is used to select the type of FIR filter (lowpass, highpass, bandpass, etc.) as well as specify the mode for frequency, gain, and whether sin(x)/x compensation is to be used.

The next screen appears as shown in Figure 7.22 where we enter the sampling rate, the band edges, and specifications for the passband ripple and stopband attenuation. The example we have chosen is a lowpass filter with a cutoff frequency of 18kHz.

FIR FILTER DESIGN LOWPASS FILTER (Screen 3)

Sampling Frequency:	*44100.0*
Passband Frequency:	*18000.0*
Stopband Frequency:	*21000.0*
Passband Ripple:	*1.00000E-02*
Stopband Ripple: (Attenuation)	*96dB*

Figure 7.22

The program will then calculate the required filter coefficients. When this calculation is complete, the screen shown in Figure 7.23 appears which lets us know the number of coefficients (taps) required to implement the filter. If the number of taps is compatible with the throughput of the DSP processor and the sampling rate, the user allows the program to proceed.

FIR DESIGN EXAMPLE (Screen 4)

Estimated Number of Taps of FIR Filter: 69

Enter Number of Taps Desired: *69*

Figure 7.23

If the 69-tap FIR filter is implemented in the ADSP-2101 microcomputer, each tap requires one processor cycle (80ns). The total processing time is therefore 5.5μs. This implies that sampling rates up to about 182kHz can be achieved and still maintain real-time operation.

ADSP-2101 PROCESSOR TIME FOR 69 TAP FIR FILTER

- 80ns (One Processor Cycle) per Tap
- 69 Taps
- 5.5μs Processor Time (80ns x 69)
- 182kHz Sampling Rate for Real-Time Operation

Figure 7.24

The next step shown in Figure 7.25 is to quantize the coefficients to the correct number of bits so that the coefficients are compatible with the DSP processor being used. In this example, the ADSP-2101 is to be used. It is a 16 bit fixed point machine, so the coefficients are quantized to 16 bits.

FIR DESIGN EXAMPLE (Screen 5)

Select the Desired Number of Bits for Quantization

Number of Bits (8 to 32): *16*

Figure 7.25

Now that the coefficients are calculated and properly quantized, we must see what effects on filter performance have been introduced by the quantization process. It should be noted that the filter design program initially calculates the coefficients with very high resolution. When these very accurate coefficients are quantized to a lower resolution, i.e. 16 bits, some accuracy is lost. This loss in accuracy may adversely affect the performance of the filter. To verify the proper performance, the filter is simulated. In this example the simulation is performed with 16 bit math. Figure 7.26 shows the simulated filter response so that the filter performance can be analyzed. Also available as outputs are the impulse response (shown in Figure 7.27) and the step response (shown

in Figure 7.28). It should be clear that this filter has no analog counterpart. The rule of thumb for calculating the required number of poles of an analog filter having this transition band characteristic (85dB from 18 to 21kHz) would indicate a 65th order filter! (Refer to Section III).

If the filter performance is satisfactory, the coefficient file can be downloaded to the DSP hardware for the filter implementation. If the response is not satisfactory, the design process may be iterated with changes made either to the number of taps or other parameters until the desired response is achieved.

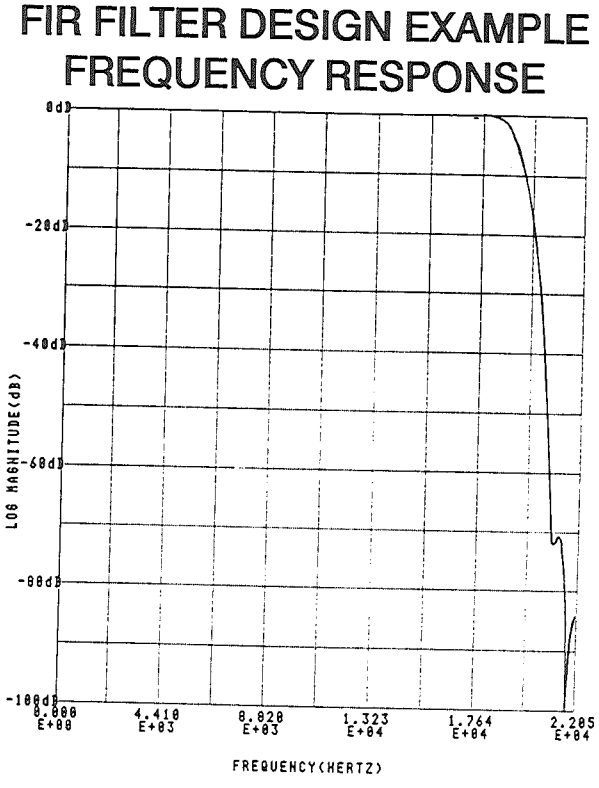


Figure 7.26

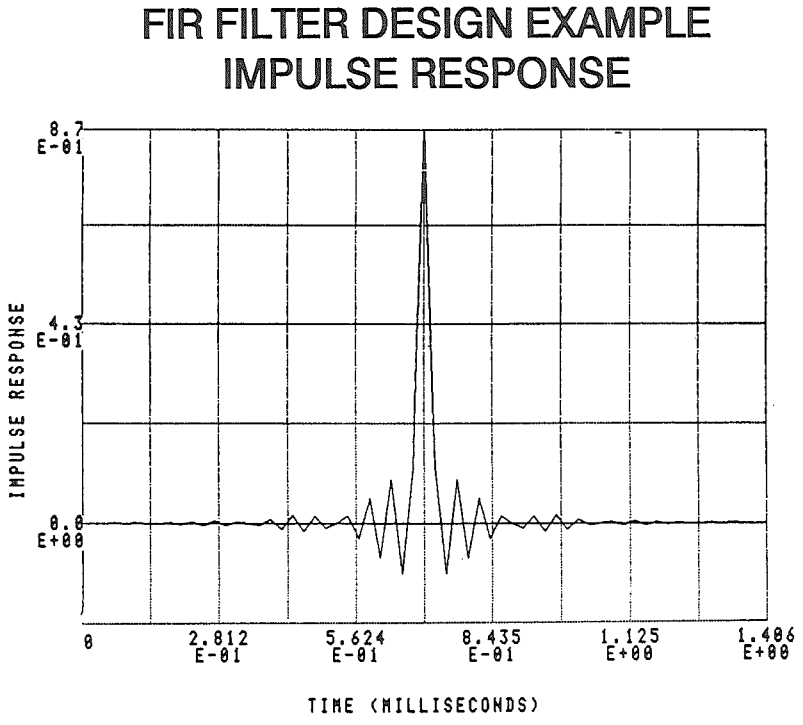


Figure 7.27

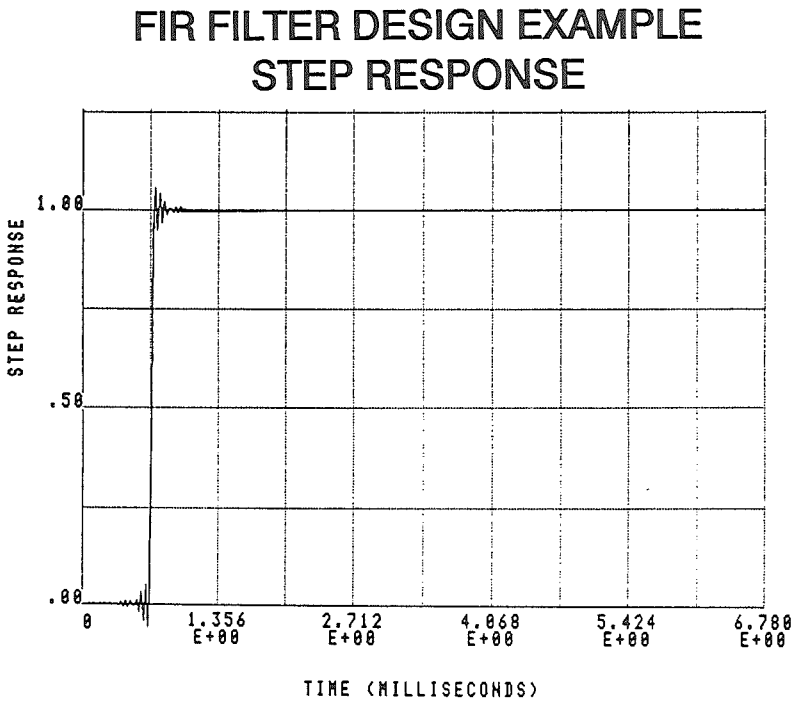


Figure 7.28

INSURING LINEAR PHASE IN FIR FILTERS

An advantage of FIR filters is they can always be made to have linear phase response which is a characteristic that makes them extremely attractive in audio and sonar applications. Linear phase means that all input frequencies are delayed by the same amount through the filter. In an FIR filter, this is the time required for the signal to propagate through the N taps. This delay is often referred to as *group delay* when applied to a band of frequencies. The group delay is constant for a linear phase FIR filter.

In order to insure phase linearity in an FIR filter, it is required that the filter coefficients are symmetric as in the case of a simple lowpass filter (Figure 7.29) or as in the case of a simple highpass filter (Figure 7.30). In addition, using an odd number of taps is also a requirement for linear phase.

SYMMETRICAL FILTER COEFFICIENTS PRODUCE LINEAR PHASE RESPONSE - LOWPASS FILTER

Figure 7.29

SYMMETRICAL FILTER COEFFICIENTS PRODUCE LINEAR PHASE RESPONSE - HIGHPASS FILTER

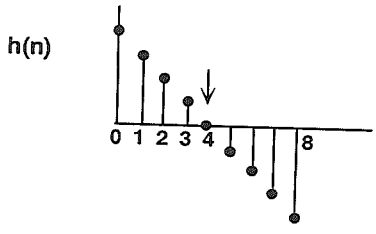


Figure 7.30

DECIMATION USING FIR FILTERS

FIR filters lend themselves to applications where data rate decimation is required, such as in oversampled sigma-delta ADCs. If we want to decimate the output data rate of an FIR filter by a factor of 2, for instance, we would take only every other sample point out of the filter. This also implies that the filter output computations need only be done every other sampling clock period. In other words, the DSP processor now has two sampling clock intervals to complete the convolution calculation. This implies that either more filter taps can be used, or perhaps a slower processor.

FIR FILTER PROPERTIES SUMMARY

- Always Stable
- Have Linear Phase, Constant Group Delay
- Can be Adaptive
- Low Round-Off Noise
- Computational Advantages When Decimating Output
- Easy to Understand and Implement

Figure 7.31

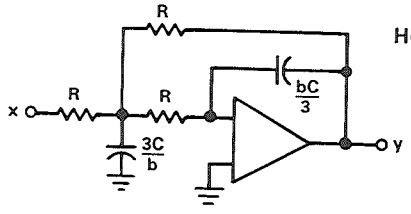
INFINITE IMPULSE RESPONSE (IIR) DIGITAL FILTERS

As was mentioned previously, digital FIR filters have no real analog counterparts, the closest analogy being the weighted moving average. In addition, FIR filters have only zeros and no poles. On the other hand, IIR filters have traditional analog counterparts (Butterworth, Chebyshev, and Elliptic) and can be analyzed and synthesized using more familiar traditional filter design techniques.

Figure 7.32 shows a second-order lowpass active filter, and its IIR digital filter equivalent is shown in Figure 7.33. This second-order IIR filter is referred to as the *biquad* (because it is described with a biquadratic equation in the z-domain) and forms the basic building block for most higher order IIR designs. The difference equation which describes the characteristics of the filter with 5 coefficients is also shown in the figure.

The general digital filter equation is shown in Figure 7.34 which gives rise to the general transfer function H(z) which contains polynomials in both the numerator and the denominator. The roots of the denominator determine the pole locations of the filter, and the roots of the numerator determine the zero locations. Although it is possible to construct a high order IIR filter directly from this equation (called the *direct form* implementation), accumulation errors due to quantization errors (finite wordlength arithmetic) may give rise to instability and large errors. For this reason, it is common to cascade several biquad sections with appropriate coefficients rather than use the direct form implementation. The biquads can be scaled separately and then cascaded in order to minimize the coefficient quantization and the recursive accumulation errors. Cascaded biquads execute more slowly than their direct form counterparts, but are more stable and minimize the effects of errors due to finite arithmetic errors. In calculating the throughput time of a particular DSP IIR filter, one should examine the benchmark performance specification for a biquad filter section. For the ADSP-2101, the execution time for a single biquad section is 560ns, corresponding to seven instruction cycles.

SECOND-ORDER ANALOG FILTER IMPLEMENTATION

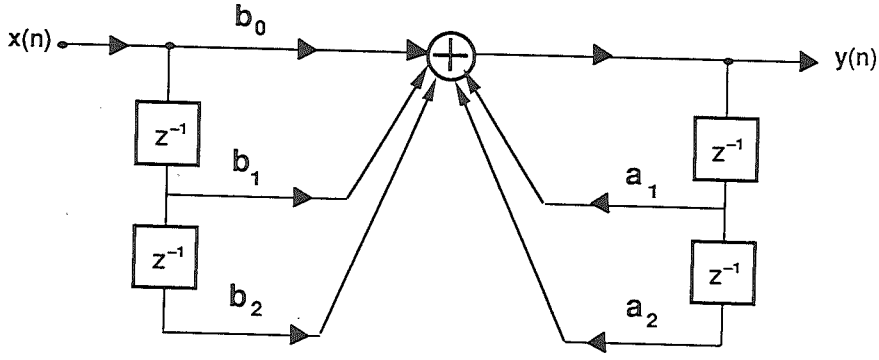


$$H(s) = \frac{Y(s)}{X(s)} = \frac{1}{1 + b(RCs) + (RCs)^2}$$

$$s = j2\pi f$$

Figure 7.32

IIR FILTER TOPOLOGY DIRECT FORM I SECOND-ORDER SECTION



$$y(n) = b_0\,x(n) + b_1\,x(n-1) + b_2\,x(n-2) - a_1\,y(n-1) - a_2\,y(n-2)$$

Figure 7.33

GENERAL FILTER EQUATION

FEEDFORWARD FEEDBACK

$$y(n) = \sum_{k=0}^{M} b_k x(n-k) + \sum_{k=1}^{N} a_k y(n-k)$$

GIVES RISE TO THE TRANSFER FUNCTION

$$H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k} \quad (\text{ZEROS})}{1 - \sum_{k=1}^{N} a_k z^{-k} \quad (\text{POLES})}$$

Figure 7.34

IIR FILTER PROPERTIES SUMMARY

- Feedback (Recursion)
- Potentially Unstable
- Usually Implemented as Cascaded Biquads Rather than Direct Form
- Non-Linear Phase
- More Efficient Than FIR Filters
- No Computational Advantage when Decimating Output
- Analogous to Analog Filters

Figure 7.35

THROUGHPUT CONSIDERATION FOR IIR FILTERS

- Determine How Many Biquad Sections are Required to Realize the Desired Filter Function
- Multiply by the Execution Time per Biquad (560ns for the ADSP-2101)
- The Result is the Minimum Sampling Period ($1/f_s$) Allowable for Real-Time Operation

Figure 7.36

SUMMARY: FIR VERSUS IIR FILTERS

Choosing between FIR and IIR filter designs can be somewhat of a challenge, but a few basic guidelines can be given. Typically, IIR filters are more efficient than FIR filters because they require less memory and fewer multiplications are needed. IIR filters can be designed based upon previous experience with analog filter designs. IIR filters may exhibit instability problems, but this is much less likely to occur if higher order filters are designed by cascading second-order systems.

On the other hand, FIR filters require more taps and computations for a given cutoff frequency response, but do exhibit linear phase characteristics. Since FIR filters operate on a finite history of data, if some data is corrupted (ADC sparkle codes, for example) the FIR filter will ring for only N-1 samples. Because of the feedback, however, an IIR filter will ring for a considerably longer period of time.

If sharp cutoff filters are needed and processing time is at a premium, IIR elliptic filters are in order. If the number of multiplies is not prohibitive, and linear phase is a requirement, then the FIR should be chosen.

IIR VERSUS FIR FILTERS

IIR FILTERS	FIR FILTERS
More Efficient	Less Efficient
Analog Equivalent	No Analog Equivalent
May be Unstable	Always Stable
Non-Linear Phase Response	Linear Phase Response
More Ringing on Glitches	Less Ringing on Glitches
CAD Design Packages Available	CAD Design Packages Available
No Efficiency Gained by Decimation	Decimation Increases Efficiency

Figure 7.37

FAST FOURIER TRANSFORMS

In many applications it is desired to process or analyze a signal in the frequency domain. In the analog world, this is easily accomplished using an analog spectrum analyzer. Mathematically, this process can be duplicated by taking the Fourier transform of the continuous-time analog signal. The Fourier transform yields the spectral content of the analog signal. In sampled data systems, however, this process must be accomplished by DSP processing of the ADC output data. Furthermore, there are two distinct differences between an analog and a digital spectral analysis. First, the output of the ADC is discrete quantized samples of the continuous input, x(t). In sampled data systems, the Discrete Fourier Transform (DFT) performs the transformation of the time domain samples into the frequency domain. In addition, the DFT must operate on a finite number of sampled data points, while the Fourier transform operates on a continuous waveform.

CONTINUOUS AND DISCRETE TIME-TO-FREQUENCY TRANSFORMATIONS

- **Fourier Transform Operates on Continuous-Time Waveforms**
- **Discrete Fourier Transform Operates on a Finite Number of Discrete Time Samples of a Waveform**

Figure 7.38

If x(n) is the sequence of N input data samples, then the DFT produces a sequence of N samples X(k) spaced equally in frequency. The DFT consists of a series of multiplications and additions where a data word is multiplied by a sinusoid value, and a number of these products are added together as shown in Figure 7.39.

THE DISCRETE FOURIER TRANSFORM (DFT) EQUATION

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nk/N}, \text{ where}$$

$$e^{-j2\pi nk/N} = \cos(2\pi nk/N) - j\sin(2\pi nk/N)$$

Figure 7.39

The DFT can be viewed as a correlation or comparison of the input signal to many sinusoids, evaluating the frequency content of the input signal. For example, a 1024 point DFT would require 1024 samples of the input signal and 1024 points from a sinusoid. Sinusoids of 1024 different frequencies equally spaced from $-f_s/2$ to $+f_s/2$ are used. Each pass of the DFT checks the sinusoid against the input signal to see how much of that frequency is present in the input signal. This is repeated for each of the 1024 frequencies. The result is shown in Figure 7.40 where N/2 discrete frequency components appear in the output spectrum. If the sampling frequency is f_s, then the spacing between the spectral lines is f_s/N, or $1/Nt_s$, where t_s is the sampling period, $1/f_s$.

Spectral analysis is most often performed with complex signals (having both real and imaginary components) so that phase information as well as amplitude and frequency information is obtained. In the above example, 1024 complex data values are multiply/accumulated with 1024 complex sinusoid values. This requires 1024 complex multiplies. This process is repeated for each of the 1024 frequencies for a total of 1024^2 multiplies, or in general terms, N^2 complex multiplies. Even for a powerful DSP device, this number of computations can be cumbersome and time consuming. This amount of computation is only required when all output frequencies are to be calculated. If the value of frequency content for only one or a few frequencies is to be determined, the computational load is not as heavy.

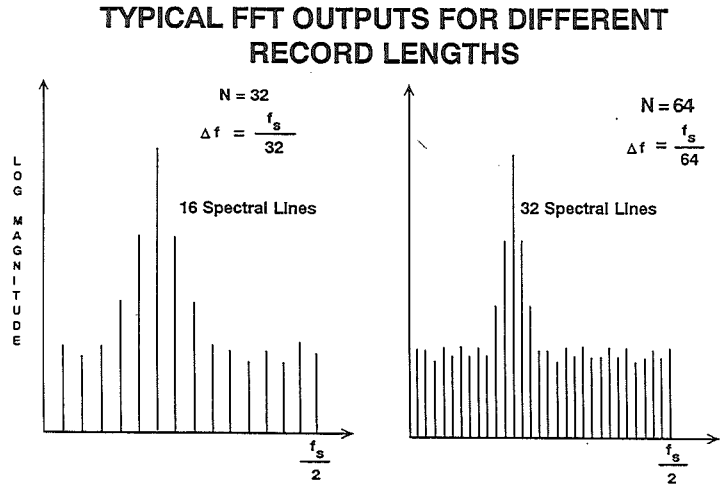


Figure 7.40

In most spectral analysis situations, however, the entire frequency spectrum up to $f_s/2$ must be computed, so we must find a faster method! The FFT is simply an algorithm to speed up the DFT calculation by reducing the number of multiplications and accumulations required. It was popularized by J. W. Cooley in the 1960s and was actually a rediscovery of an idea of Runge (1903) and Danielson and Lanczos (1942), first occurring prior to the availability of computers and calculators-when numerical calculation could take many manhours.

The FFT is based on taking advantage of certain algebraic and trigonometric symmetries in the DFT computational process. For example, if a 1024 point DFT is performed, 1024^2 (1,048,567) complex multiplications are required. It is possible to break up the 1024 point DFT into two 512 point DFTs and end up with the same results. This is called *decimation*. Each 512 point DFT requires 512^2 (262,144) complex multiplications for a total of 524,288 complex multiplications. This is a significant reduction compared to the original 1,048,567. Figure 7.41 shows an N-point DFT broken up into two N/2-point DFTs. The presence of a phase factor W (sometimes called a *twiddle factor*) on a horizontal line indicates a multiplication by W. The points where the arrows intersect the horizontal lines indicates a summation. The presence of a -1 on the line indicates a sign reversal.

If it's possible to break up the 1024 point DFT into two 512 point DFTs and still get the same result, why can't each 512 point DFT be broken up into two 256 point DFTs for an even greater reduction in computations? Well, they can. This decimation process can continue until the original DFT is broken up into 2 point DFTs (the smallest DFT possible).

The final series of computations, after the decimation process is complete, is the FFT. This is shown for the 8 point DFT in Figure 7.42. Since the FFT was first decimated by a factor of 2, the FFT is known as a Radix-2 FFT. If the *initial* DFT was decimated by a factor of 4, it would be referred to as a Radix-4 FFT. Note that the input data points are taken in normal order, but the outputs are in bit-reversed order. Bit-reversing hardware is therefore common in DSP processors such as the ADSP-2101. The basic calculation, essentially the 2 point DFT, is commonly referred to as a *butterfly* calculation. The FFT is made up of many butterfly calculations. Figure 7.43 shows the basic butterfly for the Radix-2 decimation-in-time FFT which requires one complex multiply operation per butterfly.

The significance of the FFT on the reduction in computations required to do the DFT is shown in Figure 7.44.

FIRST DECIMATION IN TIME OF 8-POINT DFT

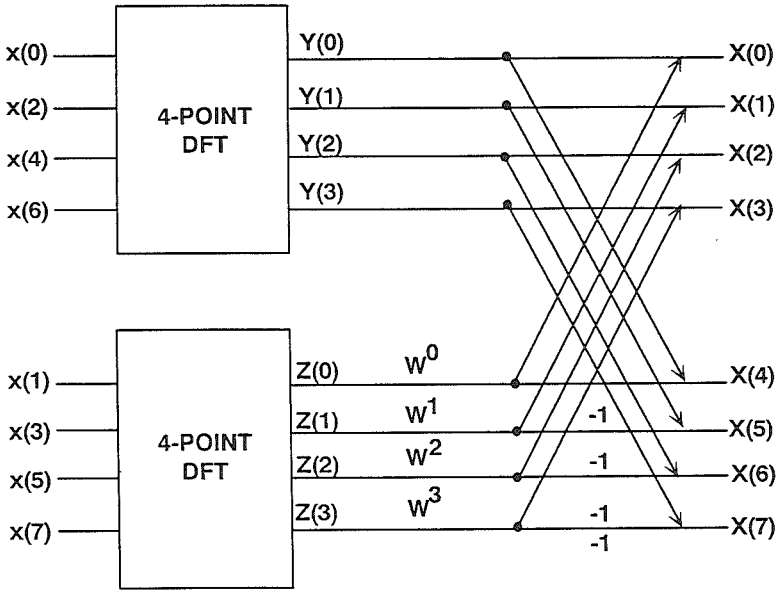


Figure 7.41

7

EIGHT-POINT DECIMATION-IN-TIME FFT NORMAL-ORDER INPUTS, BIT-REVERSED OUTPUTS

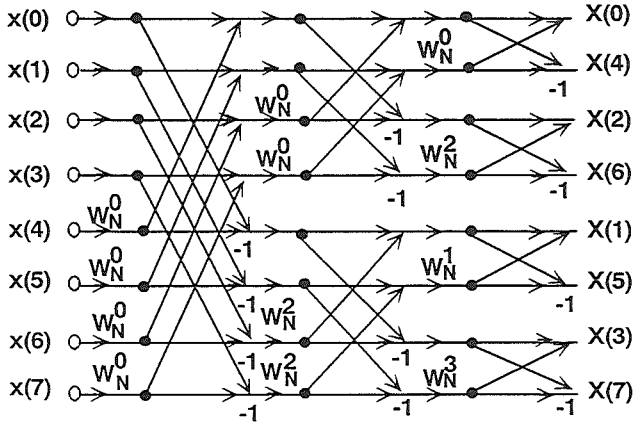
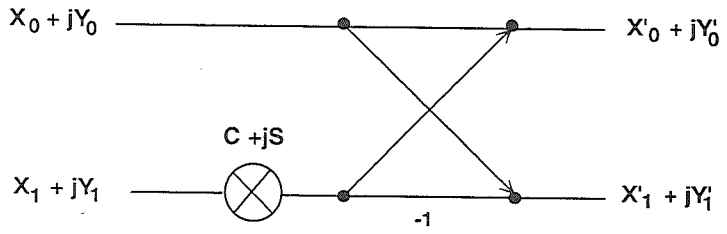


Figure 7.42

RADIX-2 DECIMATION-IN-TIME FFT BUTTERFLY



$X'_0 = X_0 + (X_1 C - Y_1 S)$ $\quad Y'_0 = Y_0 + (X_1 S - Y_1 C)$

$X'_1 = X_0 - (X_1 C - Y_1 S)$ $\quad Y'_1 = Y_0 - (X_1 S - Y_1 C)$

Figure 7.43

COMPUTATIONAL EFFICIENCY OF AN N-POINT FFT

DFT	FFT
N^2 Multiplications	$(N/2)\log_2(N)$
For N = 1024	For N = 1024
1,048,576 Multiplications	5,120 Multiplications

200:1

Figure 7.44

Note that the FFT results in the computation of *all* N/2 spectral outputs (all or nothing!). If only a few spectral points need to be calculated, the DFT is more efficient. Calculating a single spectral output using the DFT requires only N complex multiplications.

FFT HARDWARE IMPLEMENTATION

In general terms, the memory requirements for an N point FFT are N locations for real data, N locations for imaginary data, and N locations for the sinusoid data (sometimes referred to as the FFT coefficients or twiddle factors). As long as the memory requirements are met, the DSP processor must perform the necessary calculations in the required time. Many DSP vendors will either give a performance benchmark for a specified FFT size or a calculation time for a butterfly. When comparing FFT specifications, it is important to make sure that the same type of FFT is used in all cases. For example, a 1024 point FFT benchmark could have been derived from a Radix-2 or Radix-4 FFT and would not be compatible benchmarks since the number of computations required is different.

Once the basic hardware requirements are met, it is the job of the software to make the system realizable. With the same hardware, different software routines make possible a Radix-2, Radix-4, decimation-in-time or decimation-in-frequency algorithm just by manipulating the data in a different manner. An optimized Radix-4 FFT algorithm is given in Reference 6.

DSP FFT HARDWARE BENCHMARK COMPARISONS

- Radix-2, Radix-4 FFT?
- Butterfly Execution Time?
- Total FFT Execution Time?

Figure 7.45

FFT DESIGN CONSIDERATIONS

The first step in designing an FFT is to determine the number of points required, N, or the *record* length. There are several ways to approach this problem. The sampling rate, f_s, must be at least twice the maximum input signal frequency of interest. Once the sampling rate is known, the spectral resolution of the FFT is then given by f_s/N. The more points in the FFT, the better the spectral resolution. This is a prime consideration in spectral analysis applications.

In real-time analysis of speech, for example, the signal bandwidth is approximately 4kHz, implying a sampling rate of 8kHz. The spectrum of speech is not stationary. The signal must be divided up into windows, T_w, short enough to ensure that individual features are not averaged out in the FFT; all meaning is lost in the *long-term* FFT of speech, for example. But T_w must be long enough to give adequate spectral resolution. It has been determined that for human speech phonemes, 20ms is adequate, hence T_w = 20ms.

REAL-TIME SPEECH ANALYSIS FFT EXAMPLE

- BW = 4kHz, Sampling Rate = 8kHz
- Window = 20ms
- N > 8kHz x 20ms = 160, Therefore use N = 256
- Can Processor Keep Up?
- ADSP-2101 Benchmark for N=256 is 0.59ms
- Yes! With 19.41ms for Other Computations

Figure 7.46

Now, what determines if the FFT can keep up? The number of sample points in the window T_w is equal to $T_w f_s$, or 20ms x 8kHz = 160 points. This will be rounded up to the nearest power of 2, or 256 points. This says that the DSP processor must complete the 256 point FFT in less than the data acquisition time per window, T_w. Otherwise real-time processing is not possible, and the computation would have to be done off line. The

ADSP-2101 completes a 256-point FFT in 0.59ms leaving 19.41ms for other computations.

Benchmark FFT processing times for most DSP processors are given by the manufacturer. Figure 7.48 shows Radix-4 benchmark times for the ADSP-2101. The 512-point benchmark time is for a Radix-2 FFT. In evaluating various DSP processors, make sure to compare them under the same conditions. For instance, a Radix-4 FFT is somewhat faster than a Radix-2 FFT.

Figure 7.47 also shows the maximum sampling rates for real-time operation associated with the FFT execution times. These sampling rates indicate that modern DSP microcomputers such as the ADSP-2101 are capable of real-time FFT analysis of signals having bandwidths as great as 100 to 200kHz.

ADSP-2101 BENCHMARK FFT PERFORMANCE AND ASSOCIATED SAMPLING RATES FOR REALTIME OPERATION

FFT SIZE	EXECUTION TIME	MAXIMUM SAMPLING RATE
256	0.59ms	434kHz
512	1.3ms	394kHz
1024	2.9ms	353kHz
2048	6.5ms	315kHz
4096	14.2ms	288kHz

Figure 7.47

FFT OF SINEWAVE HAVING INTEGRAL NUMBER OF CYCLES IN WINDOW

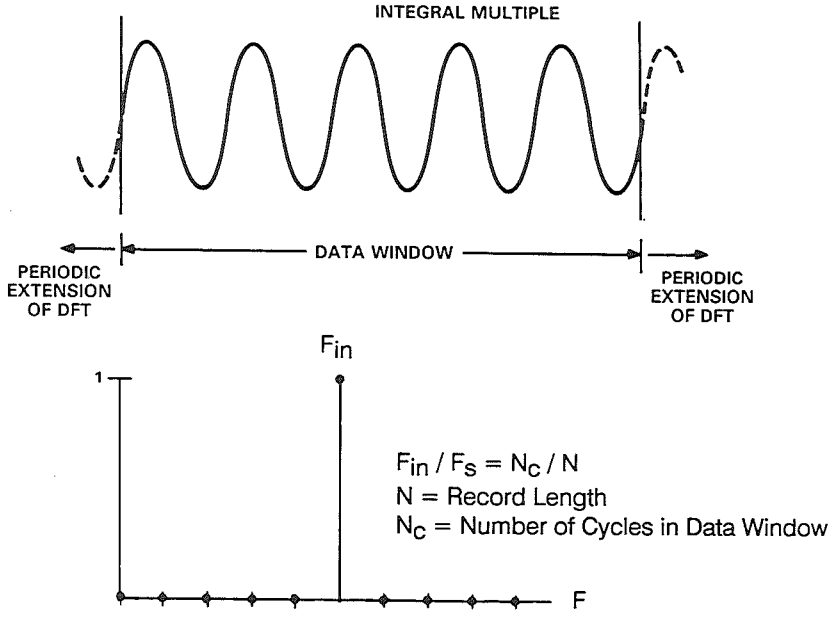


Figure 7.48

SPECTRAL LEAKAGE AND WINDOWING

Spectral leakage in FFT processing can best be understood by considering the case of performing an FFT on a pure sinewave input. Two conditions will be considered. In Figure 7.48, the ratio between the sampling rate and the input sinewave frequency is such that precisely an integral number of cycles are contained within the data window (or record length). This results in a single tone FFT spectral response at the sinewave frequency as shown in the figure. Figure 7.49 shows the condition when the sinewave does not contain an integral number of cycles within the data window. The discontinuities at the endpoints are equivalent to multiplying the sinewave by a rectangular windowing pulse which has a sin(x)/x frequency domain response. The discontinuities in the time domain result in leakage in the frequency domain, because many spectral terms are needed to fit the discontinuity. Because of the endpoint discontinuity, the FFT spectral response shows the main lobe of the sinewave being smeared, an a large number of associated sidelobes which have the basic characteristics of the rectangular time pulse.

Since in practical FFT spectral analysis applications the exact frequencies are unknown, something must be done to minimize these sidelobes. This is done by choosing a windowing function other than the rectangular window. The input time samples are multiplied by an appropriate windowing function which brings the signal to zero at the edges of the window. The selection of an appropriate windowing function is primarily a tradeoff between main-lobe spreading and sidelobe rolloff. Leakage can also be reduced by padding the data with zeros and performing a correspondingly longer FFT. Reference 4 is highly recommended for an in-depth look at windows.

The time-domain and frequency-domain characteristics of a simple windowing function (the Hanning Window) are shown in Figure 7.50. A comparison of the frequency response of the Hanning window and the more sophisticated Minimum 4-Term Blackman-Harris window is given in Figure 7.51.

7

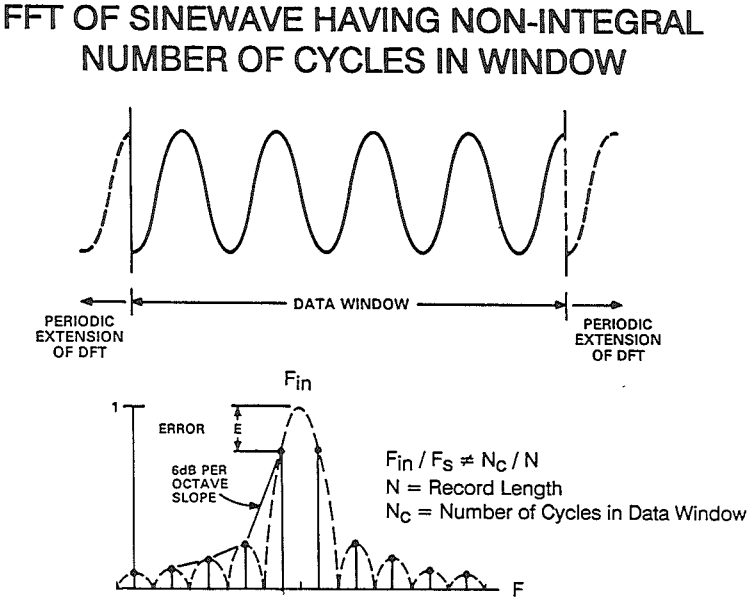


Figure 7.49

TIME AND FREQUENCY REPRESENTATION OF HANNING WINDOW

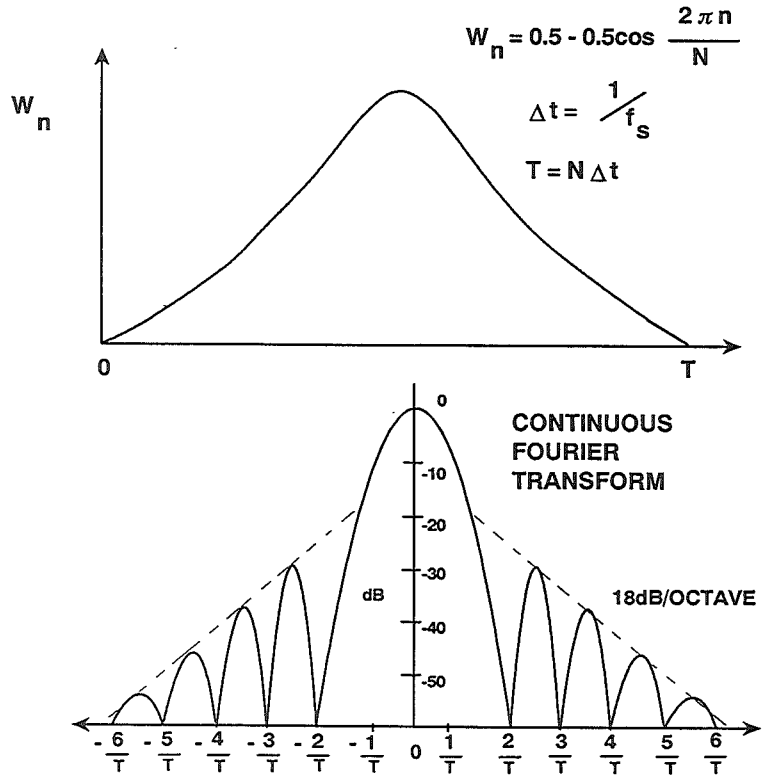


Figure 7.50

DATA SCALING AND BLOCK FLOATING POINT

The results of the butterfly calculation can be larger than the inputs to the butterfly. This data growth can pose a potential problem in a DSP with a fixed number of bits. To prevent data overflow, the data needs to be scaled before hand, leaving enough extra bits for growth. Alternatively, the data can be scaled after each pass of the FFT. The technique of scaling data after each pass of the FFT is known as *block floating point*. It is called this because the full array of data is scaled as a block regardless of whether or not each element in the block needs to be scaled. The complete block is scaled so that the relative relationship of each data word remains the same. For example, if each data word is shifted right one bit (divided by 2), the absolute values have been changed but relative to each other, the data stays the same.

COMPARISON OF WEIGHTING FUNCTIONS

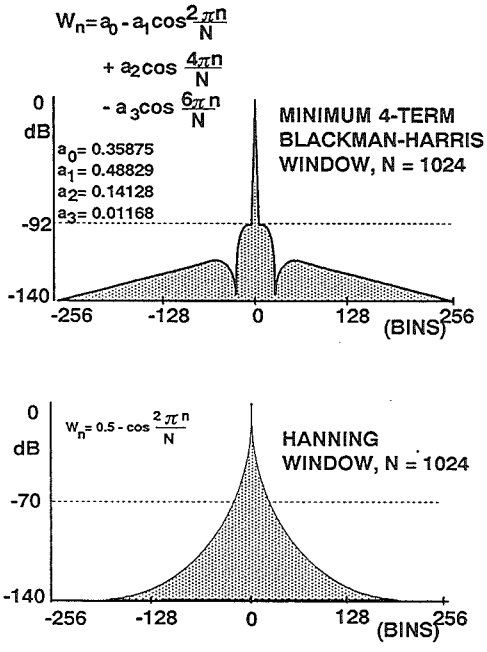


Figure 7.51

FFT SUMMARY

- The FFT is an Algorithm, not an Approximation
- Computational Speed is not Achieved at the Expense of Accuracy
- The FFT is a Fast Implementation of the DFT
- Resolution of the FFT in Frequency if f_s/N, N = Record Length
- Endpoint Discontinuities in Time Usually Require Smoothing Using Windowing Functions
- Real-Time FFT Processing Possible at Sampling Rates in Excess of 100kHz Using DSP Microcomputers

Figure 7.52

REFERENCES

1. Richard J. Higgins, **Digital Signal Processing in VLSI,** Prentice-Hall, 1990.

2. A. V. Oppenheim and R. W. Schafer, **Digital Signal Processing,** Prentice-Hall, 1975.

3. L. R. Rabiner and B. Gold, **Theory and Application of Digital Signal Processing,** Prentice-Hall, 1975.

4. Fredrick J. Harris, On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform, **Proc. IEEE,** Vol. 66, No. 1, 1978 pp. 51-83.

5. Momentum Data Systems, Inc. Costa Mesa, CA.

6. Fares Eidi, An Optimized Radix-4 Fast Fourier Transform (FFT), **Analog Devices Application Note E1329-5-9/89.** Available from Analog Devices.

7. **High Speed Design Seminar,** Analog Devices, 1990.

8. Amy Mar, Editor, **Digital Signal Processing Applications Using the ADSP-2100 Family,** Prentice-Hall, 1990.

9. C. S. Williams, **Designing Digital Filters,** Prentice-Hall, 1986.

10. R. W. Ramirez, **The FFT: Fundamentals and Concepts,** Prentice-Hall, 1985.