# SECTION 6

# DIGITAL FILTERS

- Finite Impulse Response (FIR) Filters
- Infinite Impulse Response (IIR) Filters
- Multirate Filters
- Adaptive Filters

# SECTION 6
# DIGITAL FILTERS
*Walt Kester*

## INTRODUCTION

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 in communications such as echo cancellation in modems, noise cancellation, and speech recognition.

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 amplitude and phase response are derived 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 with the sampled data values from the ADC to perform the filter calculations.

The real-time digital filter, because it is a discrete time function, works with digitized data as opposed to a continuous waveform, and a new data point is acquired each sampling period. Because of this discrete nature, data samples are referenced as numbers such as sample 1, sample 2, sample 3, etc. Figure 6.1 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 low glitch 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, $1/f_S$. A fast general purpose fixed-point DSP such as the ADSP-2189M at 75MIPS can execute a complete filter tap multiply-accumulate instruction in 13.3ns. The ADSP-2189M requires N+5 instructions for an N-tap filter. For a 100-tap filter, the total execution time is approximately 1.4μs. This corresponds to a maximum possible sampling frequency of 714kHz, thereby limiting the upper signal bandwidth to a few hundred kHz.

# DIGITAL FILTERING

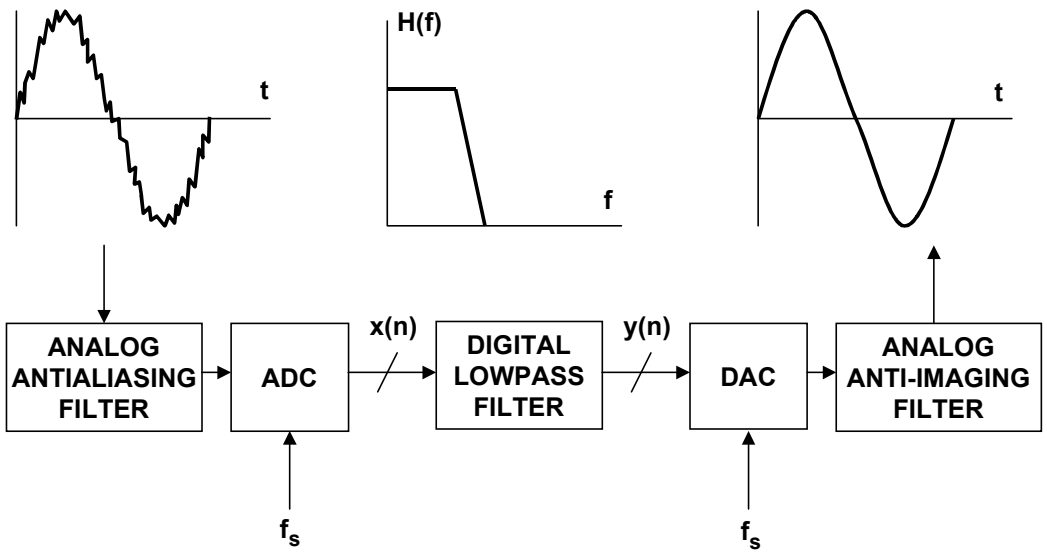



**Figure 6.1**

However, it is possible to replace a general purpose DSP chip 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-realtime operation can be maintained as in a radar system, where signal processing is typically done on bursts of data collected after each transmitted pulse.

Another option is to use a third-party dedicated DSP filter engine like the Systolix PulseDSP™ filter core. The AD7725 16-bit sigma-delta ADC has an on-chip PulseDSP filter which can do 125 million multiply-accumulates per second.

Even in highly oversampled sampled data systems, an analog antialiasing filter is still required ahead of the ADC and a reconstruction (anti-imaging) filter after the DAC. Finally, as signal frequencies increase sufficiently, they surpass the capabilities of available ADCs, and digital filtering then becomes impossible. Active analog filtering is not 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 real-time under DSP program control.

As an example, consider the comparison between an analog and a digital filter shown in Figure 6.3. The cutoff frequency of the both filters is 1kHz. The analog filter is realized as a 6-pole Chebyshev Type 1 filter (ripple in passband, no ripple in stopband). In practice, this filter would probably be realized using three 2-pole stages, each of which requires an op amp, and several resistors and capacitors. The 6-pole design is certainly not trivial, and maintaining the 0.5dB ripple specification requires accurate component selection and matching.

On the other hand, the digital FIR filter shown has only 0.002dB passband ripple, linear phase, and a much sharper roll off. In fact, it could not be realized using analog techniques! In a practical application, there are many other factors to consider when evaluating analog versus digital filters. Most modern signal processing systems use a combination of analog and digital techniques in order to accomplish the desired function and take advantage of the best of both the analog and the digital world.

## DIGITAL VERSUS ANALOG FILTERING

| DIGITAL FILTERS | ANALOG FILTERS |
|---|---|
| High Accuracy | Less Accuracy - Component Tolerances |
| Linear Phase (FIR Filters) | Non-Linear Phase |
| No Drift Due to Component Variations | Drift Due to Component Variations |
| Flexible, Adaptive Filtering Possible | Adaptive Filters Difficult |
| Easy to Simulate and Design | Difficult to Simulate and Design |
| Computation Must be Completed in Sampling Period - Limits Real Time Operation | Analog Filters Required at High Frequencies and for Anti-Aliasing Filters |
| Requires High Performance ADC, DAC & DSP | No ADC, DAC, or DSP Required |

**Figure 6.2**

There are many applications where digital filters must operate in real-time. This places specific requirements on the DSP depending upon the sampling frequency and the filter complexity. The key point is *that the DSP must finish all computations during the sampling period so it will be ready to process the next data sample*. Assume that the analog signal bandwidth to be processed is $f_a$. This requires the ADC sampling frequency $f_s$ to be at least $2f_a$. The sampling period is $1/f_s$. All DSP filter computations (including overhead) must be completed during this interval. The computation time depends on the number of taps in the filter and the speed and efficiency of the DSP. Each tap on the filter requires one multiplication and one addition (multiply-accumulate). DSPs are generally optimized to perform fast multiply-accumulates, and many DSPs have additional features such as circular buffering and zero-overhead looping minimize the "overhead" instructions that otherwise would be needed.

## ANALOG VERSUS DIGITAL FILTER FREQUENCY RESPONSE COMPARISON

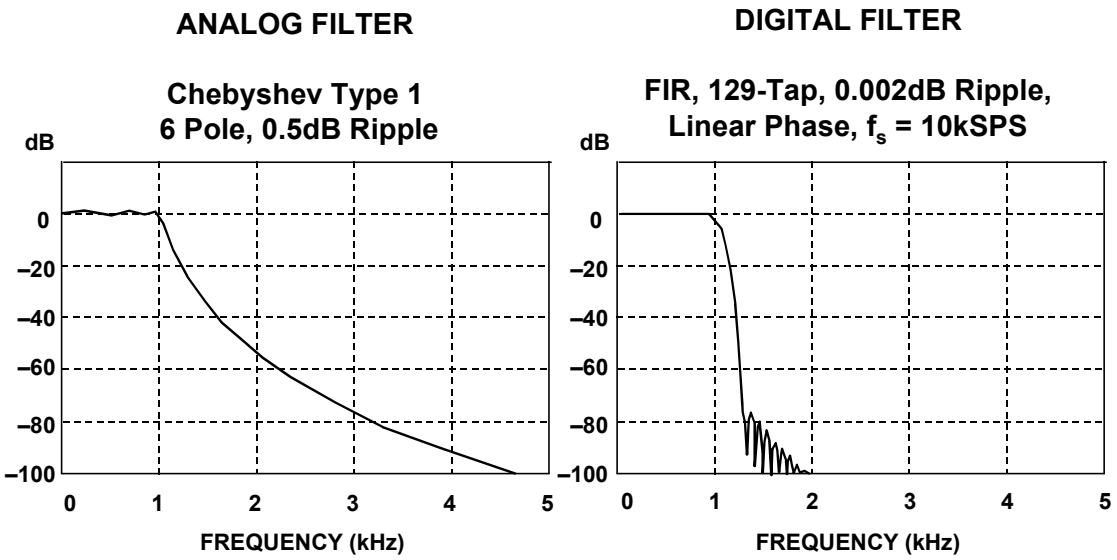



Figure 6.3

## PROCESSING REQUIREMENTS FOR REAL TIME DIGITAL FILTERING

- Signal Bandwidth = $f_a$
- Sampling Frequency $f_s > 2f_a$
- Sampling Period = $1 / f_s$
- Filter Computational Time + Overhead < Sampling Period
  - Depends on Number of Taps
  - Speed of DSP Multiplication-Accumulates (MACs)
  - Efficiency of DSP
    - Circular Buffering
    - Zero Overhead Looping
    - etc.

Figure 6.4

# FINITE IMPULSE RESPONSE (FIR) FILTERS

There are two fundamental types of digital filters: finite impulse response (FIR) and infinite impulse response (IIR). As the terminology suggests, these classifications refer to the filter's impulse response. By varying the weight of the coefficients and the number of filter taps, virtually any frequency response characteristic can be realized with an FIR filter. As has been shown, FIR filters can achieve performance levels which are not possible with analog filter techniques (such as perfect linear phase response). However, high performance FIR filters generally require a large number of multiply-accumulates and therefore require fast and efficient DSPs. On the other hand, IIR filters tend to mimic the performance of traditional analog filters and make use of feedback. Therefore their impulse response extends over an infinite period of time. Because of feedback, IIR filters can be implemented with fewer coefficients than for an FIR filter. Lattice filters are simply another way to implement either FIR or IIR filters and are often used in speech processing applications. Finally, digital filters lend themselves to adaptive filtering applications simply because of the speed and ease with which the filter characteristics can be changed by varying the filter coefficients.

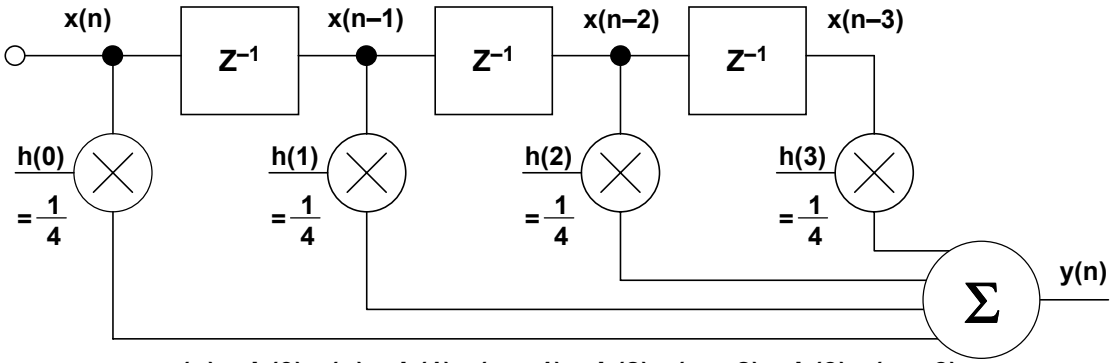
## TYPES OF DIGITAL FILTERS

- **Moving Average**
- **Finite Impulse Response (FIR)**
  - **Linear Phase**
  - **Easy to Design**
  - **Computationally Intensive**
- **Infinite Impulse Response (IIR)**
  - **Based on Classical Analog Filters**
  - **Computationally Efficient**
- **Lattice Filters (Can be FIR or IIR)**
- **Adaptive Filters**

**Figure 6.5**

The most elementary form of an FIR filter is a *moving average* filter as shown in Figure 6.6. Moving average filters are popular for smoothing data, such as in the analysis of stock prices, etc. The input samples, x(n) are passed through a series of buffer registers (labeled $z^{-1}$, corresponding to the z-transform representation of a delay element). In the example shown, there are four taps corresponding to a four-point moving average. Each sample is multiplied by 0.25, and these results are added to yield the final moving average output y(n). The figure also shows the general equation of a moving average filter with N taps. Note again that N refers to the number of filter taps, and not the ADC or DAC resolution as in previous sections.

## 4-POINT MOVING AVERAGE FILTER




$$y(n) = h(0)\,x(n) + h(1)\,x(n-1) + h(2)\,x(n-2) + h(3)\,x(n-3)$$

$$= \frac{1}{4}\,x(n) + \frac{1}{4}\,x(n-1) + \frac{1}{4}\,x(n-2) + \frac{1}{4}\,x(n-3)$$

$$= \frac{1}{4}\left[x(n) + x(n-1) + x(n-2) + x(n-3)\right]$$

**For N-Point Moving Average Filter:** $y(n) = \frac{1}{N}\sum_{k=0}^{N-1} x(n-k)$

**Figure 6.6**

Since the coefficients are equal, an easier way to perform a moving average filter is shown in Figure 6.7. Note that the first step is store the first four samples, x(0), x(1), x(2), x(3) in a register. These quantities are added and then multiplied by 0.25 to yield the first output, y(3). Note that the initial outputs y(0), y(1), and y(2) are not valid because all registers are not full until sample x(3) is received.

When sample x(4) is received, it is added to the result, and sample x(0) is subtracted from the result. The new result must then be multiplied by 0.25. Therefore, the calculations required to produce a new output consist of one addition, one subtraction, and one multiplication, regardless of the length of the moving average filter.

The step function response of a 4-point moving average filter is shown in Figure 6.8. Notice that the moving average filter has no overshoot. This makes it useful in signal processing applications where random white noise must be filtered but pulse response preserved. Of all the possible linear filters that could be used, the moving average produces the lowest noise for a given edge sharpness. This is illustrated in Figure 6.9, where the noise level becomes lower as the number of taps are increased. Notice that the 0% to 100% risetime of the pulse response is equal to the total number of taps in the filter multiplied by the sampling period.

## CALCULATING OUTPUT OF 4-POINT MOVING AVERAGE FILTER

$$y(3) = 0.25\,[\,x(3) + x(2) + x(1) + x(0)\,]$$

$$y(4) = 0.25\,[\,x(4) + x(3) + x(2) + x(1)\,]$$

$$y(5) = 0.25\,[\,x(5) + x(4) + x(3) + x(2)\,]$$

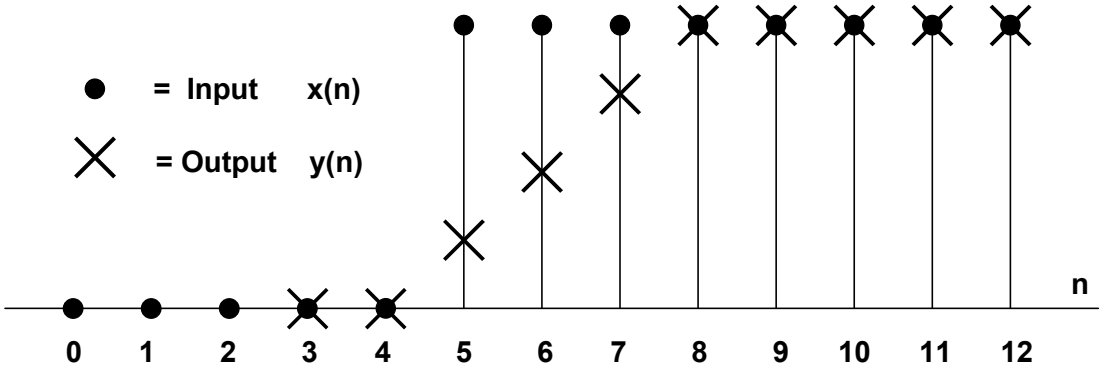
$$y(6) = 0.25\,[\,x(6) + x(5) + x(4) + x(3)\,]$$

$$y(7) = 0.25\,[\,x(7) + x(6) + x(5) + x(4)\,]$$

Each Output Requires:
1 multiplication, 1 addition, 1 subtraction

Figure 6.7

## 4-TAP MOVING AVERAGE FILTER STEP RESPONSE




General: $y(n) = \frac{1}{N} \sum_{k=0}^{N-1} x(n-k)$

For N = 4: $y(n) = \frac{1}{4} \sum_{k=0}^{3} x(n-k)$

Figure 6.8

## MOVING AVERAGE FILTER RESPONSE TO NOISE SUPERIMPOSED ON STEP INPUT

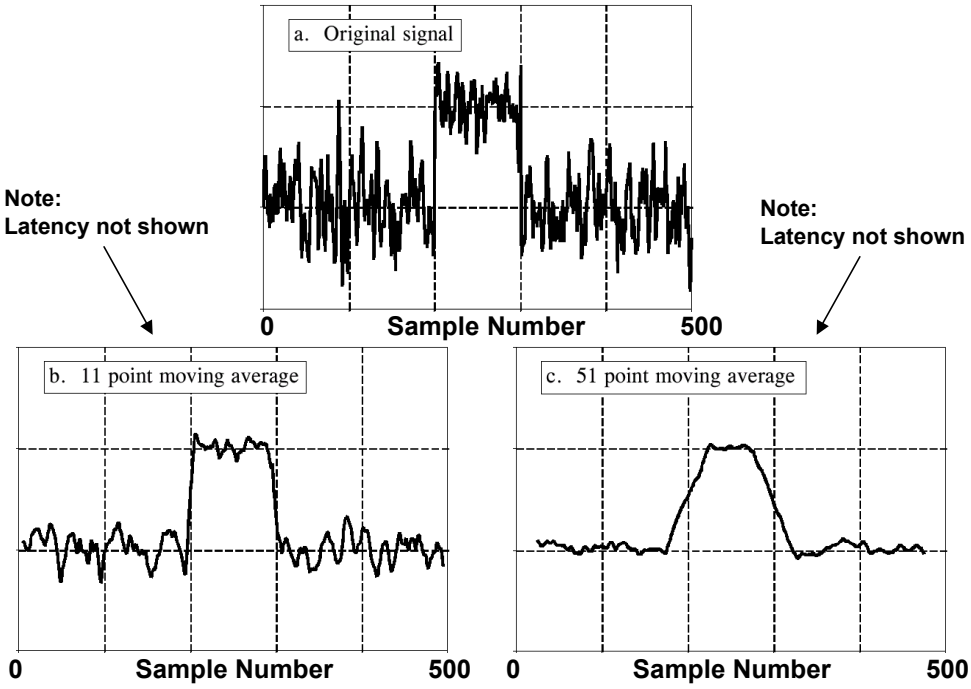



**Figure 6.9**

The frequency response of the simple moving average filter is sin(x)/x and is shown on a linear amplitude scale in Figure 6.10. Adding more taps to the filter sharpens the rolloff, but does not significantly reduce the amplitude of the sidelobes which are approximately 14dB down for the 11 and 31-tap filter. These filters are definitely not suitable where high stopband attenuation is required.

## MOVING AVERAGE FILTER FREQUENCY RESPONSE

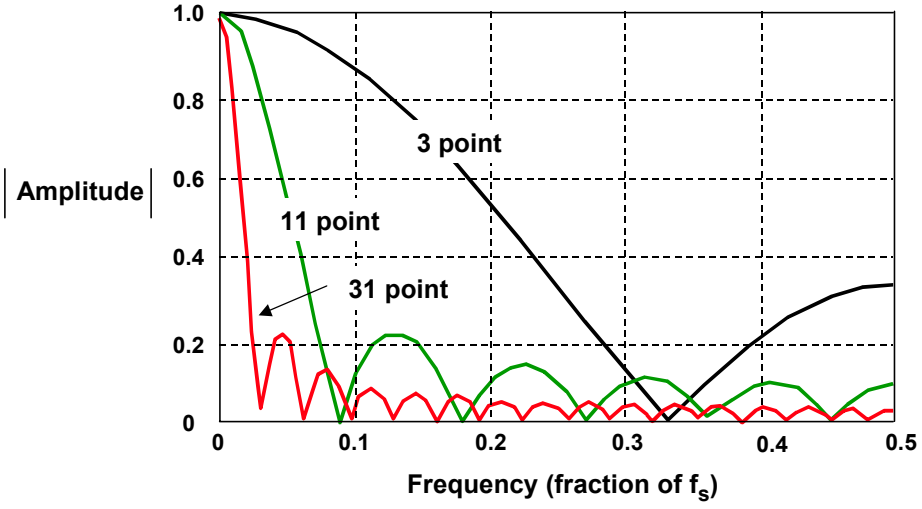
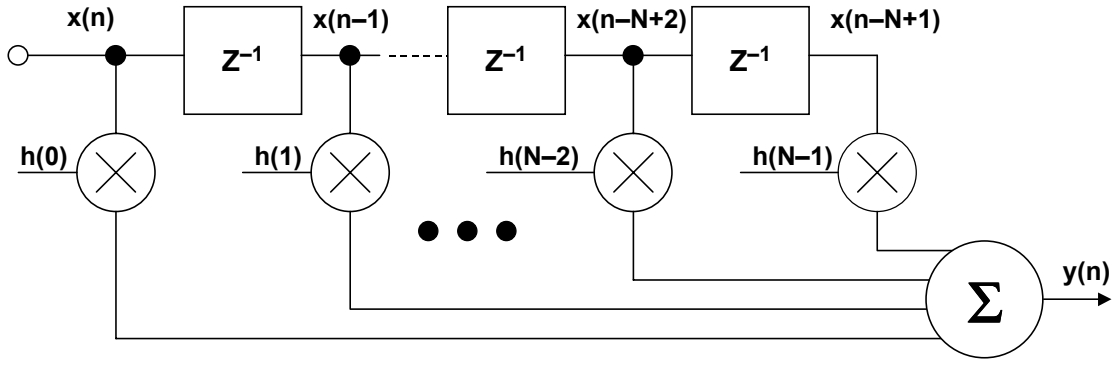



**Figure 6.10**

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. Note that unlike the moving average filter, one multiply-accumulate cycle is now required per tap for the generalized FIR filter. 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 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 discrete Fourier transform of H(f).

## N-TAP FINITE IMPULSE RESPONSE (FIR) FILTER




- $y(n) = h(n) * x(n) = \sum_{k=0}^{N-1} h(k)\, x(n-k)$
- $*$ = Symbol for Convolution
- Requires N multiply-accumulates for each output

**Figure 6.11**

The generalized form of an N-tap FIR filter is shown in Figure 6.11. As has been discussed, an FIR filter must perform the following convolution equation:

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

where h(k) is the filter coefficient array and x(n-k) 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. An N-tap FIR filter requires N multiply-accumulate cycles.

FIR filter diagrams are often simplified as shown in Figure 6.12. The summations are represented by arrows pointing into the dots, and the multiplications are indicated by placing the h(k) coefficients next to the arrows on the lines. The $z^{-1}$ delay element is often shown by placing the label above or next to the appropriate line.

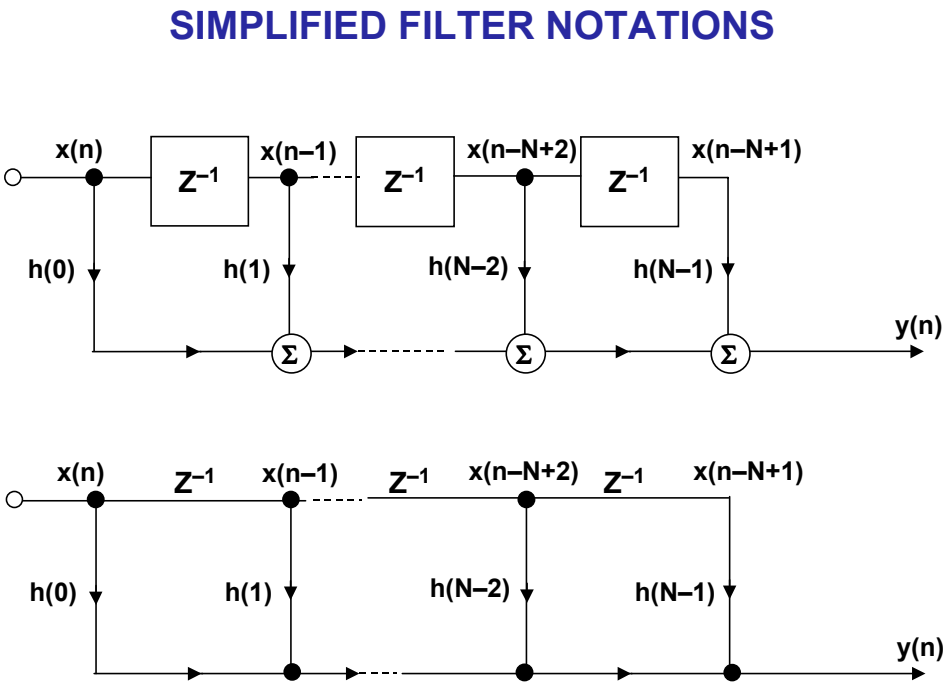


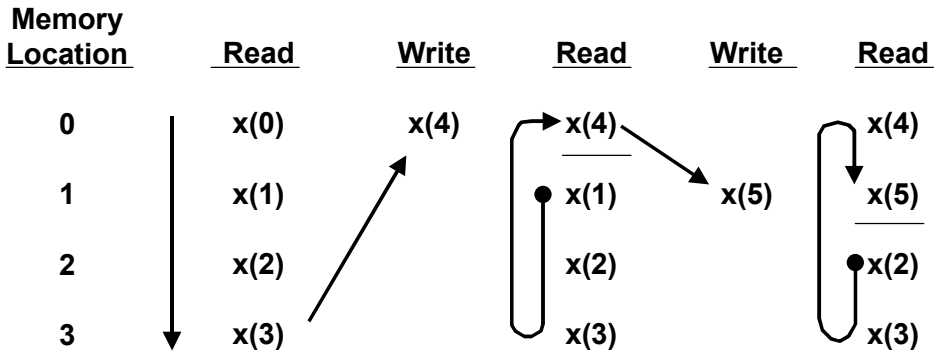

**Figure 6.12**

## FIR FILTER IMPLEMENTATION IN DSP HARDWARE USING CIRCULAR BUFFERING

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 6.13 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 always stored in RAM.

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 is reset to the beginning of the buffer.

## CALCULATING OUTPUTS OF 4-TAP FIR FILTER USING A CIRCULAR BUFFER




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

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

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

**Figure 6.13**

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 6.13, the coefficients are stored in a reverse manner.

A simple summary flowchart for these operations is shown in Figure 6.14. For Analog Devices' DSPs, *all operations within the filter loop are completed in one instruction cycle*, thereby greatly increasing efficiency. This is referred to as *zero-overhead looping*. The actual FIR filter assembly code for the ADSP-21XX family of fixed point DSPs is shown in Figure 6.15. The arrows in the diagram point to the actual executable instructions, and the rest of the code are simply comments added for clarification.

The first instruction (labeled *fir:*) sets up the computation by clearing the MR register and loading the MX0 and MY0 registers with the first data and coefficient values from data and program memory. The multiply-accumulate with dual data fetch in the *convolution* loop is then executed N–1 times in N cycles to compute the sum of the first N–1 products. The final multiply-accumulate instruction is performed with the rounding mode enabled to round the result to the upper 24 bits of the MR register. The MR1 register is then conditionally saturated to its most positive or negative value based on the status of the overflow flag contained in the MV register. In this manner, results are accumulated to the full 40-bit precision of the MR register, with saturation of the output only if the final result overflowed beyond the least significant 32 bits of the MR register.

The limit on the number of filter taps attainable for a real-time implementation of the FIR filter subroutine is determined primarily by the processor cycle time, the sampling rate, and the number of other computations required. The FIR subroutine presented here requires a total of N+5 cycles for a filter of length N. For the ADSP-2189M 75MIPS DSP, one instruction cycle is 13.3ns, so a 100-tap filter would require 13.3ns×100 + 5×13.3ns = 1330ns + 66.5ns = 1396.5ns = 1.4µs.

## PSEUDOCODE FOR FIR FILTER PROGRAM USING A DSP WITH CIRCULAR BUFFERING

1. Obtain sample from ADC (typically interrupt driven)
2. Move sample into input signal's circular buffer
3. Update the pointer for the input signal's circular buffer
4. Zero the accumulator
5. Implement filter (control the loop through each of the coefficients)
    6. Fetch the coefficient from the coefficient's circular buffer
    7. Update the pointer for the coefficient's circular buffer
    8. Fetch the sample from the input signal's circular buffer
    9. Update the pointer for the input signal's circular buffer
    10. Multiply the coefficient by the sample
    11. Add the product to the accumulator
12. Move the filtered sample to the DAC

```
ADSP-21xx Example code:

CNTR = N-1;
DO convolution UNTIL CE;
convolution:
   MR = MR + MX0 * MY0(SS), MX0 = DM(I0,M1), MY0 = PM(I4,M5);
```

**Figure 6.14**

## ADSP-21XX FIR FILTER ASSEMBLY CODE (SINGLE PRECISION)

```
.MODULE           fir_sub;
{                 FIR Filter Subroutine
                  Calling Parameters
                           I0 --> Oldest input data value in delay line
                           I4 --> Beginning of filter coefficient table
                           L0 = Filter length (N)
                           L4 = Filter length (N)
                           M1,M5 = 1
                           CNTR = Filter length - 1 (N-1)
                  Return Values
                           MR1 = Sum of products (rounded and saturated)
                           I0 --> Oldest input data value in delay line
                           I4 --> Beginning of filter coefficient table
                  Altered Registers
                           MX0,MY0,MR
                  Computation Time
                            (N - 1) + 6 cycles = N + 5 cycles
                  All coefficients are assumed to be in 1.15 format. }
.ENTRY            fir;
fir:              MR=0, MX0=DM(I0,M1), MY0=PM(I4,M5);
                  CNTR = N-1;
                  DO convolution UNTIL CE;
convolution:        MR=MR+MX0*MY0(SS), MX0=DM(I0,M1), MY0=PM(I4,M5);
                  MR=MR+MX0*MY0(RND);
                  IF MV SAT MR;
                  RTS;
.ENDMOD;
```

**Figure 6.15**

## DESIGNING FIR FILTERS

FIR filters are relatively easy to design using modern CAD tools. Figure 6.16 summarizes the characteristics of FIR filters as well as the most popular design techniques. *The fundamental concept of FIR filter design is that the filter frequency response is determined by the impulse response, and the quantized impulse response and the filter coefficients are identical.* This can be understood by examining Figure 6.17. The input to the FIR filter is an impulse, and as the impulse propagates through the delay elements, the filter output is identical to the filter coefficients. The FIR filter design process therefore consists of determining the impulse response from the desired frequency response, and then quantizing the impulse response to generate the filter coefficients.

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 multiply-accumulates. 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 6.18. 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.

## CHARACTERISTICS OF FIR FILTERS

- Impulse Response has a Finite Duration (N Cycles)
- Linear Phase, Constant Group Delay (N Must be Odd)
- No Analog Equivalent
- Unconditionally Stable
- Can be Adaptive
- Computational Advantages when Decimating Output
- Easy to Understand and Design
  - Windowed-Sinc Method
  - Fourier Series Expansion with Windowing
  - Frequency Sampling Using Inverse FFT - Arbitrary Frequency Response
  - Parks-McClellan Program with Remez Exchange Algorithm

Figure 6.16

## FIR FILTER IMPULSE RESPONSE DETERMINES THE FILTER COEFFICIENTS

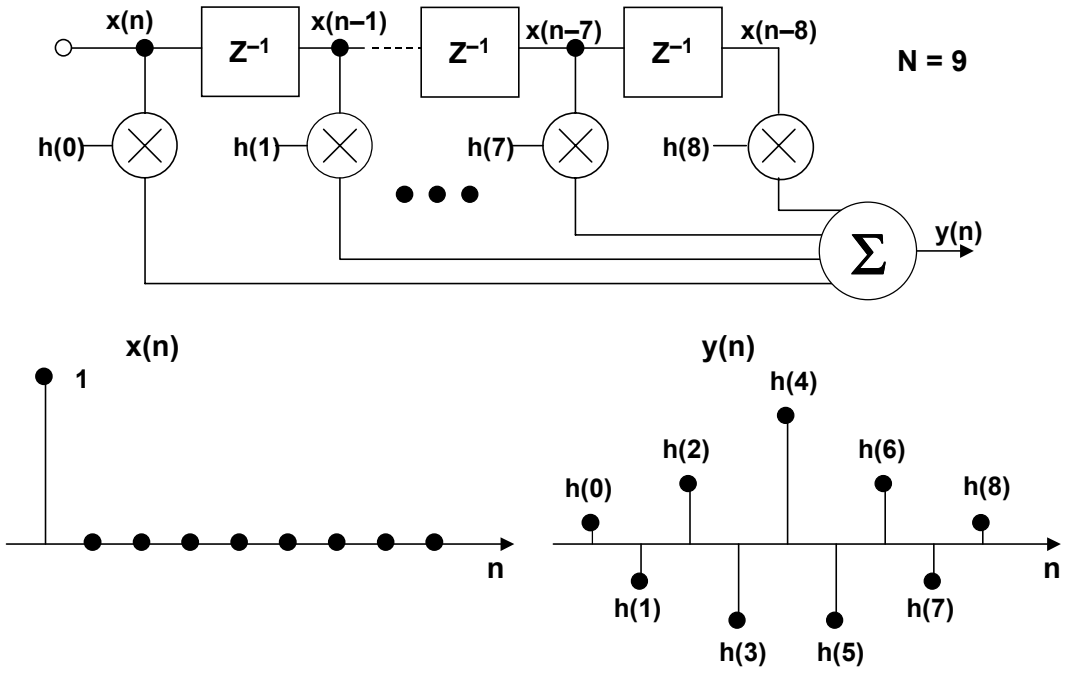



Figure 6.17

## DUALITY OF TIME AND FREQUENCY

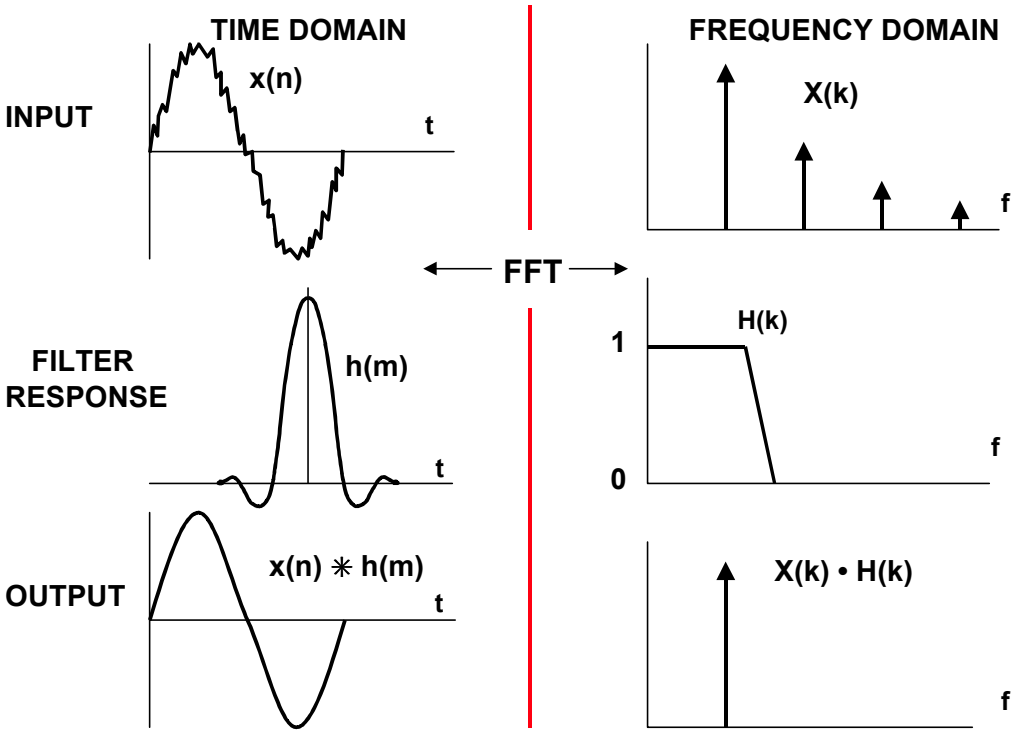



**Figure 6.18**

The transfer function in the frequency domain (either a 1 or a 0) can be translated to the time domain by the discrete Fourier transform (in practice, the fast Fourier transform is used). 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 desired impulse response are the FIR filter coefficients.

The mathematics involved in filter design (analog or digital) generally 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, 3, 4, 5, and 6, but the theory is not necessary for the rest of this discussion.

### FIR Filter Design Using the Windowed-Sinc Method

An ideal lowpass filter frequency response is shown in Figure 6.19A. The corresponding impulse response in the time domain is shown in Figure 6.19B, and follows the sin(x)/x (sinc) function. If an FIR filter is used to implement this frequency response, an infinite number of taps are required. The windowed-sinc method is used to implement the filter as follows. First, the impulse response is truncated to a reasonable number of N taps as in Figure 6.19C. As has been discussed in Section 5, the frequency response corresponding to Figure 6.19C has relatively poor sidelobe performance because of the end-point discontinuities in the truncated impulse response. The next step in the design process is to apply an appropriate window function as shown in Figure 6.19D to the truncated impulse. This forces the endpoints to zero. The particular window function chosen determines the rolloff and sidelobe performance of the filter. Window functions have been discussed in detail in Section 5, and there are several good choices depending upon the desired frequency response. The frequency response of the truncated and windowed-sinc impulse response of Figure 6.19E is shown in Figure 6.19F.

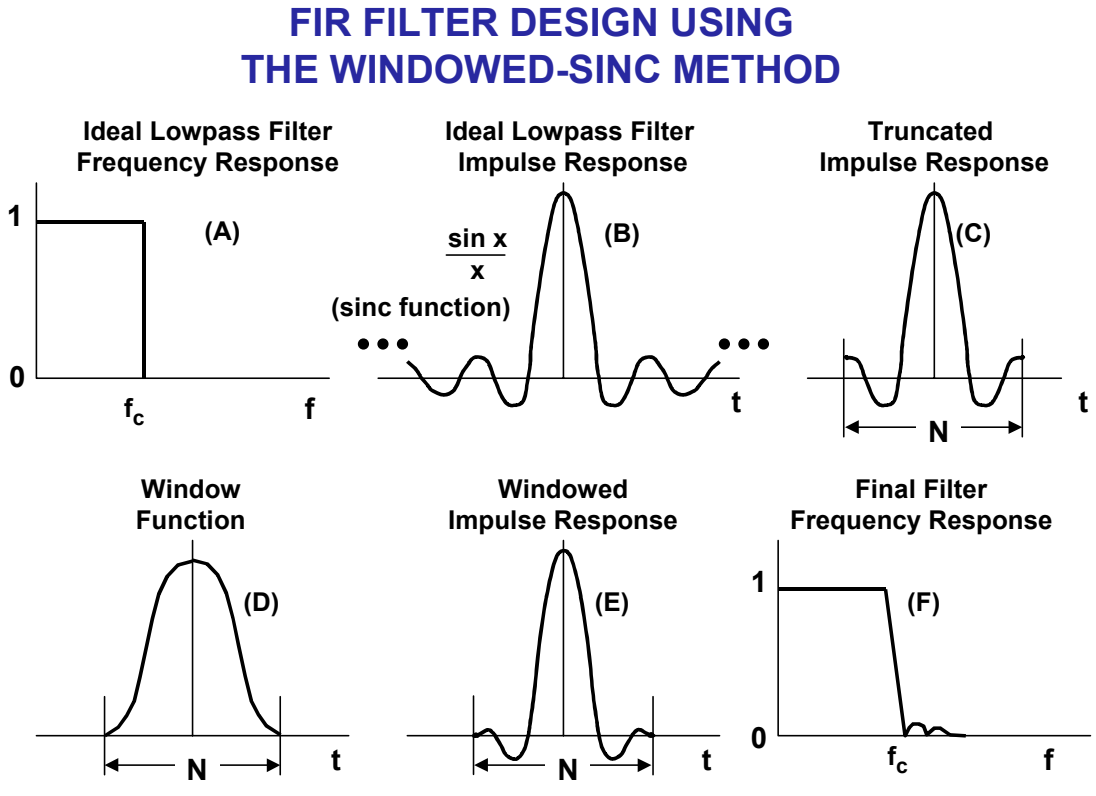



Figure 6.19

### FIR Filter Design Using the Fourier Series Method with Windowing

The Fourier series with windowing method (Figure 6.20) starts by defining the transfer function H(f) mathematically and expanding it in a Fourier series. The Fourier series coefficients define the impulse response and therefore the coefficients of the FIR filter. However, the impulse response must be truncated and windowed as in the previous method. After truncation and windowing, an FFT is used to generate the corresponding frequency response. The frequency response can be modified by choosing different window functions, although precise control of the stopband characteristics is difficult in any method which uses windowing.

## FIR FILTER DESIGN USING FOURIER SERIES METHOD WITH WINDOWING

- **Specify H(f)**
- **Expand H(f) in a Fourier series: The Fourier series coefficients are the coefficients of the FIR filter, h(m), and its Impulse Response**
- **Truncate the Impulse Response to N points (taps)**
- **Apply a suitable Window function to h(m) to smooth the effects of truncation**
- **Lacks precise control of cutoff frequency; Highly dependent on Window function**

**Figure 6.20**

### FIR Filter Design Using the Frequency Sampling Method

This method is extremely useful in generating an FIR filter with an arbitrary frequency response. H(f) is specified as a series of amplitude and phase points in the frequency domain. The points are then converted into real and imaginary components. Next, the impulse response is obtained by taking the complex inverse FFT of the frequency response. The impulse response is then truncated to N points, and a window function is applied to minimize the effects of truncation. The filter design should then be tested by taking its FFT and evaluating the frequency response. Several iterations may be required to achieve the desired response.

## FREQUENCY SAMPLING METHOD FOR FIR FILTERS WITH ARBITRARY FREQUENCY RESPONSE

- **Specify H(k) as a Finite Number of Spectral Points Spread Uniformly Between 0 and 0.5$f_s$ (512 Usually Sufficient)**
- **Specify Phase Points (Can Make Equal to Zero)**
- **Convert Rectangular Form (Real + Imaginary)**
- **Take the Complex Inverse FFT of H(k) Array to Obtain the Impulse Response**
- **Truncate the Impulse Response to N Points**
- **Apply a suitable Window function to h(m) to smooth the effects of truncation**
- **Test Filter Design and Modify if Necessary**
- **CAD Design Techniques More Suitable for Lowpass, Highpass, Bandpass, or Bandstop Filters**

**Figure 6.21**

### FIR Filter Design Using the Parks-McClellan Program

Historically, the design method based on the use of windows to truncate the impulse response and to obtain the desired frequency response was the first method used for designing FIR filters. The frequency-sampling method was developed in the 1970s and is still popular where the frequency response is an arbitrary function.

Modern CAD programs are available today which greatly simplify the design of lowpass, highpass, bandpass, or bandstop FIR filters. A popular one was developed by Parks and McClellan and uses the Remez exchange algorithm. The filter design begins by specifying the parameters shown in Figure 6.22: passband ripple, stopband ripple (same as attenuation), and the transition region. For this design example, the QED1000 program from Momentum Data Systems was used (a demo version is free and downloadable from http://www.mds.com).

For this example, we will design an audio lowpass filter that operates at a sampling rate of 44.1kHz. The filter is specified as shown in Figure 6.22: 18kHz passband frequency, 21kHz stopband frequency, 0.01dB passband ripple, 96dB stopband ripple (attenuation). We must also specify the wordlength of the coefficients, which in this case is 16 bits, assuming a 16-bit fixed-point DSP is to be used.

## FIR CAD TECHNIQUES: PARKS McCLELLAN PROGRAM WITH REMEZ EXCHANGE ALGORITHM

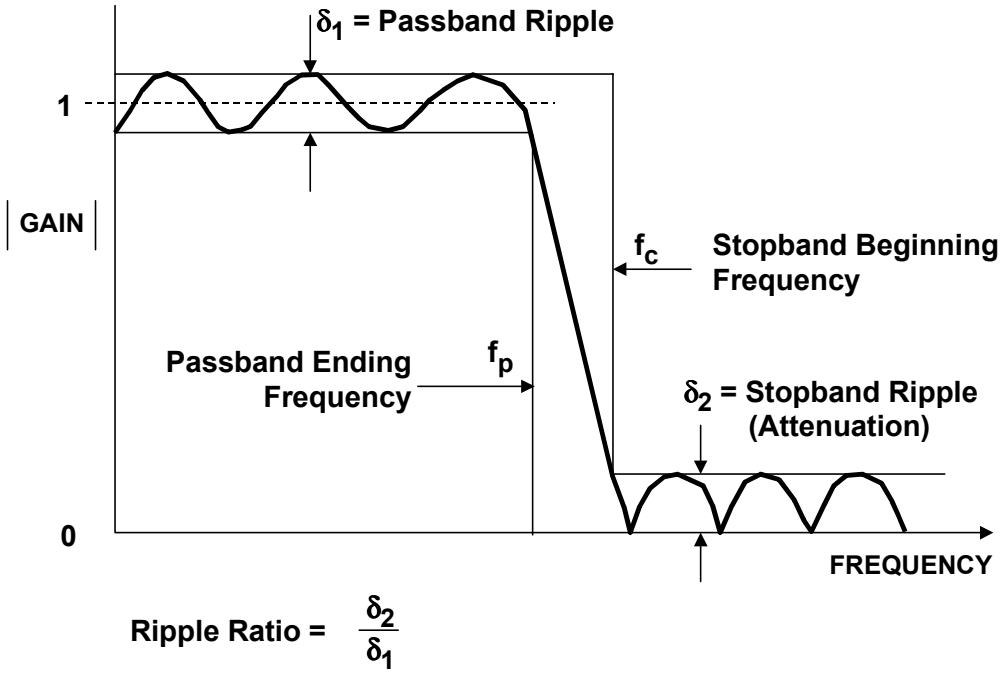



Figure 6.22

## PARKS McCLELLAN EQUIRIPPLE FIR FILTER DESIGN: PROGRAM INPUTS

- Filter Type:
  - Lowpass
  - Highpass
  - Bandpass
  - Bandstop
  - Differentiator
  - Multiband
- Sampling Frequency: 44,100Hz
- Passband Frequency: 18,000Hz
- Stopband Frequency: 21,000Hz
- Passband Ripple: 0.01dB
- Stopband Ripple (Attenuation): 96dB
- Wordlength: 16-bits

Figure 6.23

The program allows us to choose between a window-based design or the equiripple Parks-McClellan program. We will choose the latter. The program now estimates the number of taps required to implement the filter based on the above specifications. In this case, it is 69 taps. At this point, we can accept this and proceed with the design or decrease the number of taps and see what degradation is specifications occur.

We will accept this number and let the program complete the calculations. The program outputs the frequency response (Figure 6.25), step function response (Figure 6.26), s and z-plane analysis data, and the impulse response (Figure 6.27). The QED1000 program then outputs the quantized filter coefficients to a program which generates the actual DSP assembly code for a number of popular DSPs, including Analog Devices'. The program is quite flexible and allows the user to perform a number of scenarios to optimize the filter design.

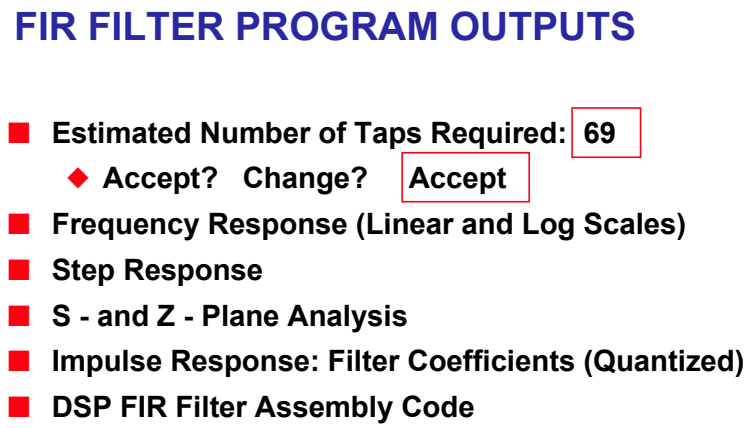



**Figure 6.24**

The 69-tap FIR filter requires 69 + 5 = 74 instruction cycles using the ADSP-2189M 75MIPS processor, which yields a total computation time per sample of 74×13.3ns = 984ns. The sampling interval is 1/44.1kHz, or 22.7µs. This allows 22.7µs – 0.984µs = 21.7µs for overhead and other operations.

Other options are to use a slower processor for this application (3.3MIPS), a more complex filter which takes more computation time (up to N = 1700), or increase the sampling frequency to about 1MSPS.

# FIR DESIGN EXAMPLE: FREQUENCY RESPONSE

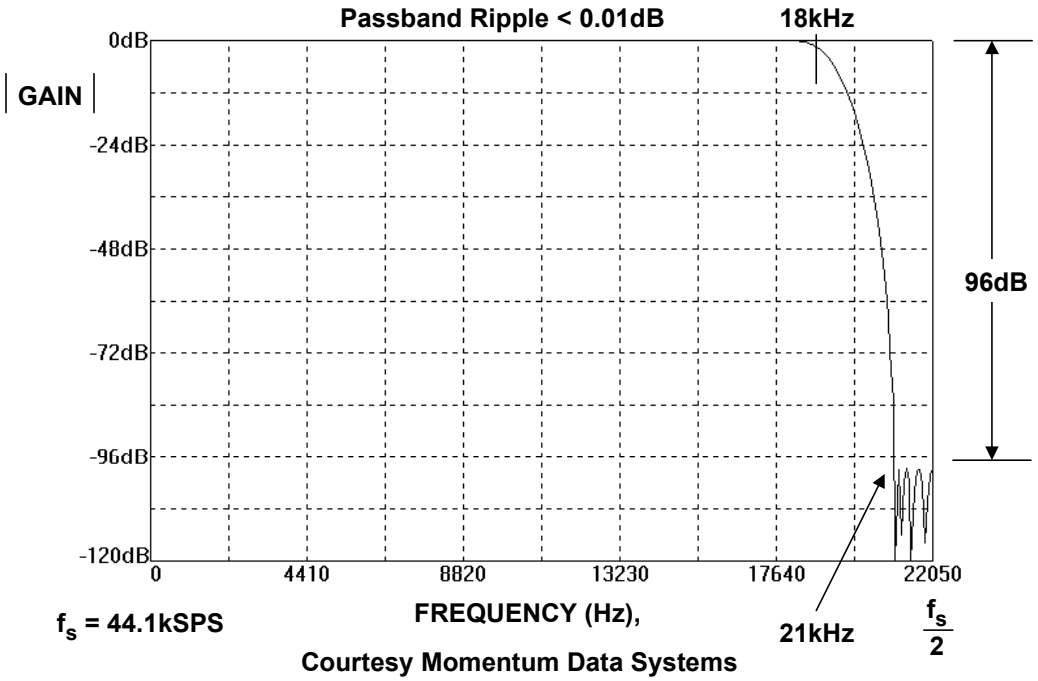



**Figure 6.25**

# FIR FILTER DESIGN EXAMPLE: STEP RESPONSE

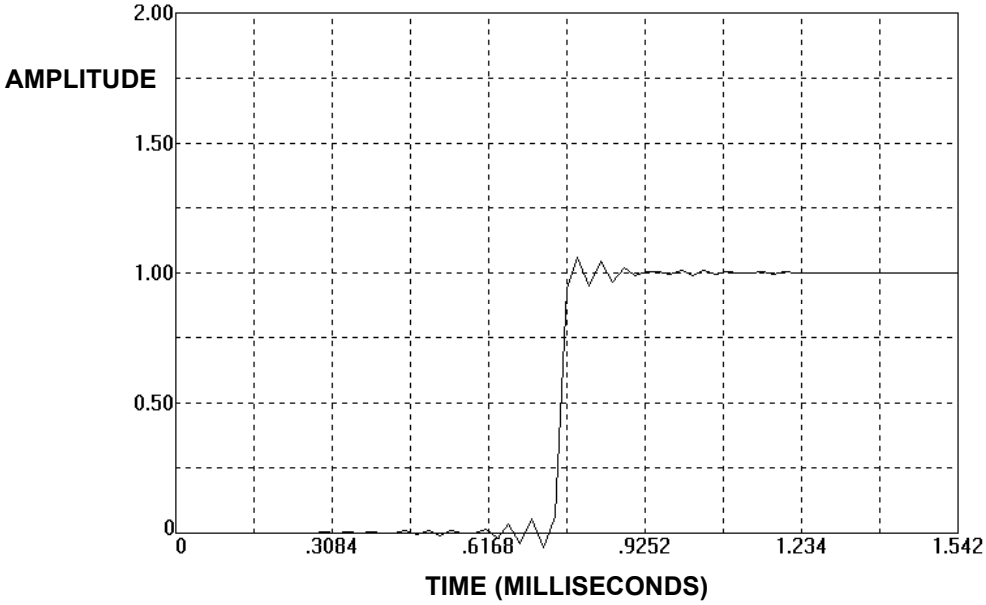
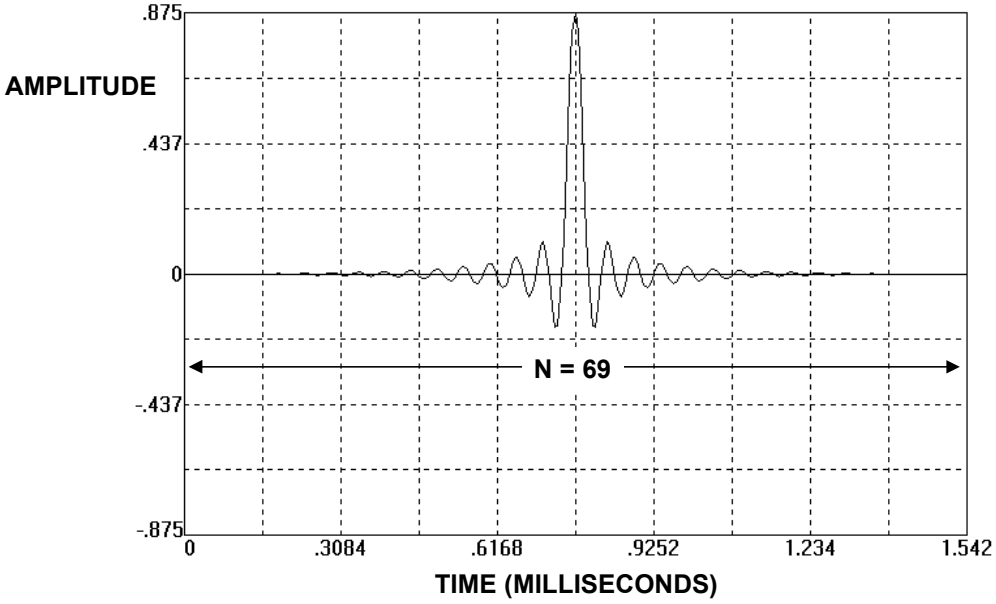



**Figure 6.26**

# FIR DESIGN EXAMPLE: IMPULSE RESPONSE (FILTER COEFFICIENTS)




Courtesy Momentum Data Systems

Figure 6.27

# DESIGN EXAMPLE USING ADSP-2189M: PROCESSOR TIME FOR 69-TAP FIR FILTER

- Sampling Frequency $f_S$ = 44.1kSPS
- Sampling Interval = 1 / $f_S$ = 22.7µs
- Number of Filter Taps, N = 69
- Number of Required Instructions = N + 5 = 74
- Processing Time / Instruction = 13.3ns (75MIPS) (ADSP-2189M)
- Total Processing Time = 74×13.3ns = 984ns
- Total Processing Time < Sampling Interval with 22.7µs – 0.984µs = 21.7µs for Other Operations
  - Increase Sampling Frequency to 1MHz
  - Use Slower DSP (3.3MIPS)
  - Add More Filter Taps (Up to N = 1700)

Figure 6.28

### Designing Highpass, Bandpass, and Bandstop Filters Based on Lowpass Filter Design

Converting a lowpass filter design impulse response into a highpass filter impulse response can be accomplished in one of two ways. In the *spectral inversion method*, the sign of each filter coefficient in the lowpass filter impulse response is changed. Next, 1 is added to the center coefficient. In the *spectral reversal method*, the sign of every other coefficient is changed. This reverses the frequency domain plot. In other words, if the cutoff of the lowpass filter is $0.2f_S$, the resulting highpass filter will have a cutoff frequency of $0.5f_S - 0.2f_S = 0.3f_S$. This must be considered when doing the original lowpass filter design.

## DESIGNING HIGHPASS FILTERS USING LOWPASS FILTER IMPULSE RESPONSE

- **Spectral Inversion Technique:**
  - **Design Lowpass Filter (Linear Phase, N odd)**
  - **Change the Sign of Each Coefficient in the Impulse Response, h(m)**
  - **Add 1 to the Coefficient at the Center of Symmetry**

- **Spectral Reversal Technique:**
  - **Design Lowpass Filter**
  - **Change the Sign of Every Other Coefficient in the Impulse Response, h(m)**
  - **This Reverses the Frequency Domain left-for-right:**
    **0 becomes 0.5, and 0.5 becomes 0;**
    **i.e., if the cutoff frequency of the lowpass filter is 0.2, the cutoff of the resulting highpass filter is 0.3**

**Figure 6.29**

Bandpass and bandstop filters can be designed by combining individual lowpass and highpass filters in the proper manner. Bandpass filters are designed by placing the lowpass and highpass filters in cascade. The equivalent impulse response of the cascaded filters is then obtained by *convolving* the two individual impulse responses.

A bandstop filter is designed by connecting the lowpass and highpass filters in parallel and adding their outputs. The equivalent impulse response is then obtained by *adding* the two individual impulse responses.

## BANDPASS AND BANDSTOP FILTERS DESIGNED FROM LOWPASS AND HIGHPASS FILTERS

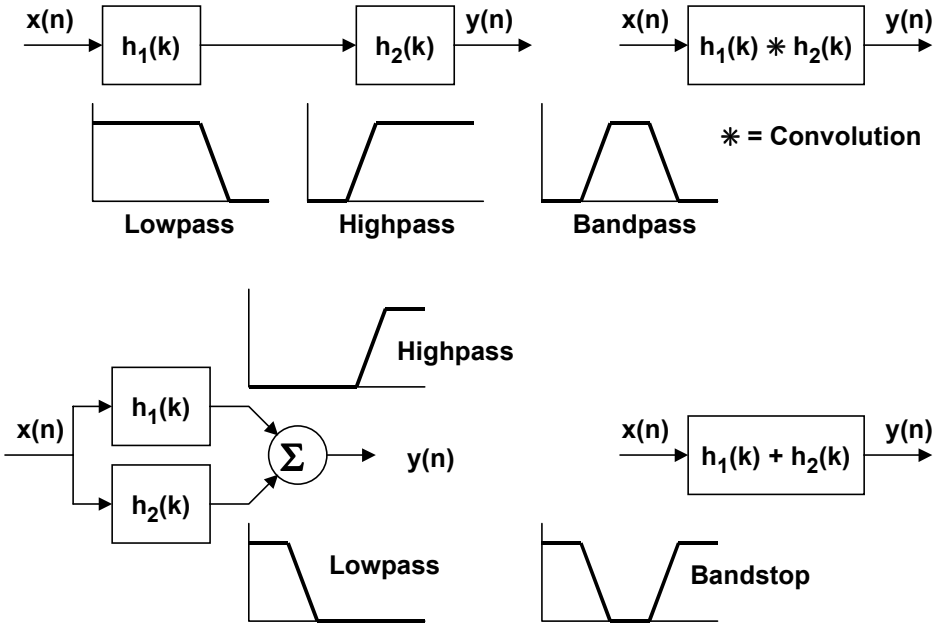



**Figure 6.30**

# INFINITE IMPULSE RESPONSE (IIR) FILTERS

As was mentioned previously, 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, Elliptic, and Bessel) and can be analyzed and synthesized using more familiar traditional filter design techniques.

Infinite impulse response filters get their name because their impulse response extends for an infinite period of time. This is because they are recursive, i.e., they utilize feedback. Although they can be implemented with fewer computations than FIR filters, IIR filters do not match the performance achievable with FIR filters, and do not have linear phase. Also, there is no computational advantage achieved when the output of an IIR filter is decimated because each output value must always be calculated.

# INFINITE IMPULSE RESPONSE (IIR) FILTERS

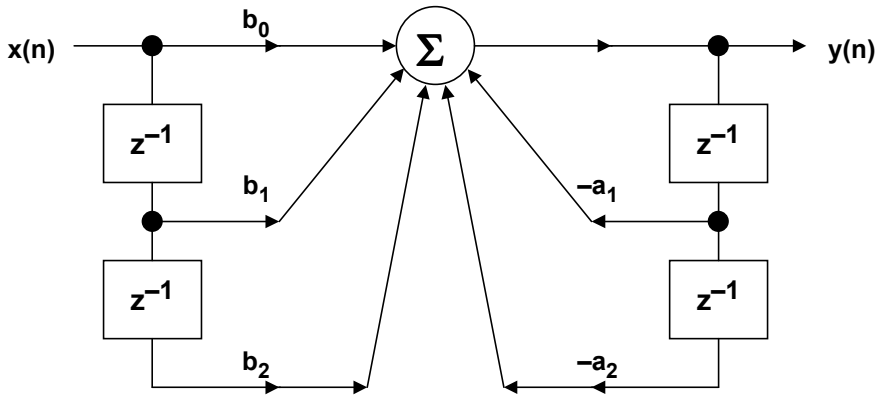
- Uses Feedback (Recursion)
- Impulse Response has an Infinite Duration
- Potentially Unstable
- Non-Linear Phase
- More Efficient than FIR Filters
- No Computational Advantage when Decimating Output
- Usually Designed to Duplicate Analog Filter Response
- Usually Implemented as Cascaded Second-Order Sections (Biquads)

**Figure 6.31**

IIR filters are generally implemented in two-pole sections called biquads because they are described with a biquadratic equation in the z-domain. Higher order filters are designed using cascaded biquad sections, i.e., a 6-pole filter requires 3 biquad sections.

The basic IIR biquad is shown in Figure 6.32. The zeros are formed by the feedforward coefficients $b_0$, $b_1$, and $b_2$; the poles are formed by the feedback coefficients $a_1$, and $a_2$.

## HARDWARE IMPLEMENTATION OF SECOND-ORDER IIR FILTER (BIQUAD) DIRECT FORM 1



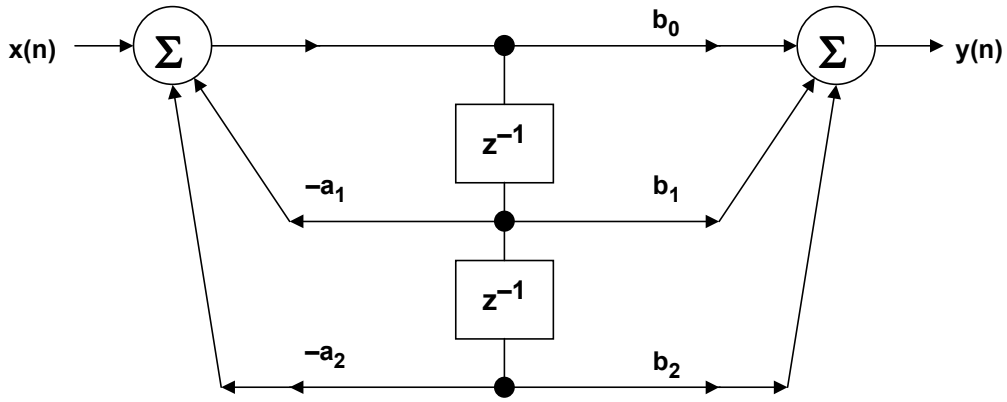

- $y(n) = b_0x(n) + b_1x(n–1) + b_2x(n–2) – a_1y(n–1) – a_2y(n–2)$
- $y(n) = \sum_{k=0}^{M} b_k x(n–k) – \sum_{k=1}^{N} a_k y(n–k)$
- $H(z) = \dfrac{\sum_{k=0}^{M} b_k z^{-k}}{1 + \sum_{k=1}^{N} a_k z^{-k}}$ (Zeros) / (Poles)

**Figure 6.32**

The general digital filter equation is shown in Figure 6.32 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.

The Direct Form 1 biquad section shown in Figure 6.32 requires four registers. This configuration can be changed into an equivalent circuit shown in Figure 6.33 which is called the Direct Form 2 which requires only two registers. It can be shown that the equations describing the Direct Form 2 IIR biquad filter are the same as those for Direct Form 1. As in the case of FIR filters, the notation for an IIR filter is often simplified as shown in Figure 6.34.

## IIR BIQUAD FILTER DIRECT FORM 2




- Reduces to the same equation as Direct Form 1:
- $y(n) = b_0x(n) + b_1x(n–1) + b_2x(n–2) – a_1y(n–1) – a_2y(n–2)$
- Requires Only 2 Delay Elements (Registers)

**Figure 6.33**

## IIR BIQUAD FILTER SIMPLIFIED NOTATIONS

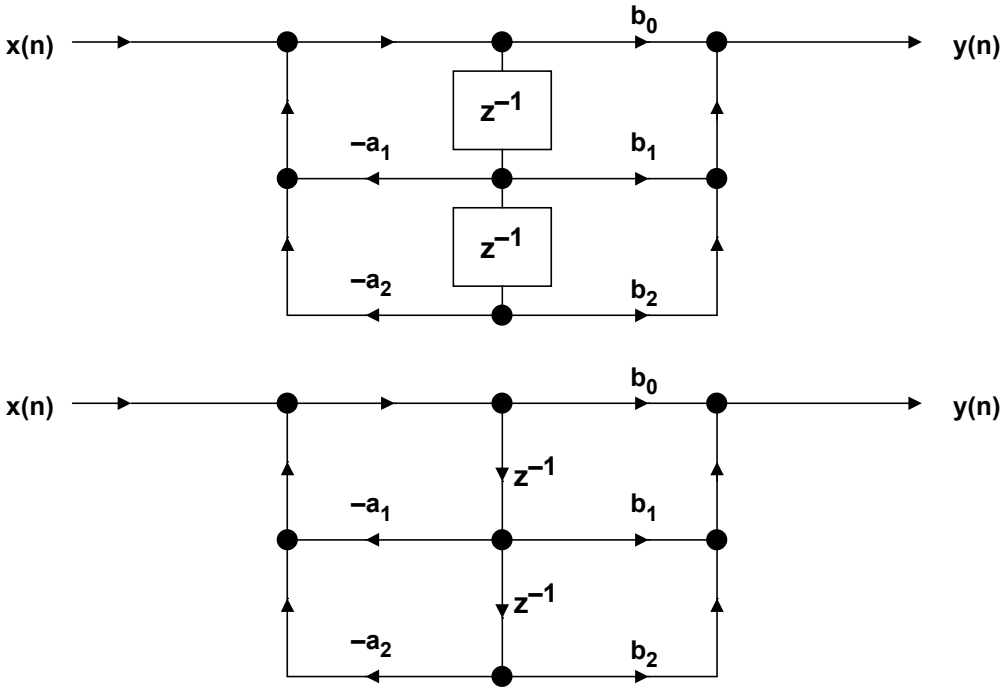



**Figure 6.34**

# IIR FILTER DESIGN TECHNIQUES

A popular method for IIR filter design is to first design the analog-equivalent filter and then mathematically transform the transfer function H(s) into the z-domain, H(z). Multiple pole designs are implemented using cascaded biquad sections. The most popular analog filters are the Butterworth, Chebyshev, Elliptical, and Bessel (see Figure 6.35). There are many CAD programs available to generate the Laplace transform, H(s), for these filters.

The all-pole Butterworth (also called maximally flat) has no ripple in the passband or stopband and has monotonic response in both regions. The all-pole Type 1 Chebyshev filter has a faster rolloff than the Butterworth (for the same number of poles) and has ripple in the passband. The Type 2 Chebyschev filter is rarely used, but has ripple in the stopband rather than the passband.

The Elliptical (Cauer) filter has poles and zeros and ripple in both the passband and stopband. This filter has even faster rolloff than the Chebyshev for the same number of poles. The Elliptical filter is often used where degraded phase response can be tolerated.

Finally, the Bessel (Thompson) filter is an all-pole filter which is optimized for pulse response and linear phase but has the poorest rolloff of any of the types discussed for the same number of poles.

## REVIEW OF POPULAR ANALOG FILTERS

- **Butterworth**
  - **All Pole, No Ripples in Passband or Stopband**
  - **Maximally Flat Response (Fastest Roll-off with No Ripple)**
- **Chebyshev (Type 1)**
  - **All Pole, Ripple in Passband, No Ripple in Stopband**
  - **Shorter Transition Region than Butterworth for Given Number of Poles**
  - **Type 2 has Ripple in Stopband, No Ripple in Passband**
- **Elliptical (Cauer)**
  - **Has Poles and Zeros, Ripple in Both Passband and Stopband**
  - **Shorter Transition Region than Chebyshev for Given Number of Poles**
  - **Degraded Phase Response**
- **Bessel (Thompson)**
  - **All Pole, No Ripples in Passband or Stopband**
  - **Optimized for Linear Phase and Pulse Response**
  - **Longest Transition Region of All for Given Number of Poles**

**Figure 6.35**

All of the above types of analog filters are covered in the literature, and their Laplace transforms, H(s), are readily available – either from tables or CAD programs. There are three methods used to convert the Laplace transform into the z-transform: *impulse invariant* transformation, *bilinear* transformation, and the *matched z-transform*. The resulting z-transforms can be converted into the coefficients of the IIR biquad. These techniques are highly mathematically intensive and will not be discussed further.

A CAD approach for IIR filter design is similar to the Parks-McClellan program used for FIR filters. This technique uses the Fletcher-Powell algorithm.

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-21xx-family, seven instruction cycles are required to execute a biquad filter output sample. For the ADSP-2189M, 75MIPS DSP, this corresponds to 7 × 13.3ns = 93ns, corresponding to a maximum possible sampling frequency of 10MSPS (neglecting overhead).

## IIR FILTER DESIGN TECHNIQUES

- Impulse Invarient Transformation Method
  - Start with H(s) for Analog Filter
  - Take Inverse Laplace Transform to get Impulse Response
  - Obtain z-Transform H(z) from Sampled Impulse Response
  - z-Transform Yields Filter Coefficients
  - Aliasing Effects Must be Considered
- Bilinear Transformation Method
  - Another Method for Transforming H(s) into H(z)
  - Performance Determined by the Analog System's Differential Equation
  - Aliasing Effects do not Occur
- Matched z-Transform Method
  - Maps H(s) into H(z) for filters with both poles and zeros
- CAD Methods
  - Fletcher-Powell Algorithm
  - Implements Cascaded Biquad Sections

Figure 6.36

## THROUGHPUT CONSIDERATIONS FOR IIR FILTERS

- Determine How Many Biquad Sections (N) are Required to Realize the Desired Frequency Response
- Multiply this by the number of instruction cycles per Biquad for the DSP and add overhead cycles (5N + 2 cycles for the ADSP-21xx series, for example).
- The Result (plus overhead) is the Minimum Allowable Sampling Period ($1 / f_S$) for Real-Time Operation

Figure 6.37

### 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 multiply-accumulates 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 multiply-accumulates for a given cutoff frequency response, but have 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 a good choice. If the number of multiply/accumulates is not prohibitive, and linear phase is a requirement, then the FIR should be chosen.

## COMPARISON BETWEEN FIR AND IIR 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 6.38**

# MULTIRATE FILTERS

There are many applications where it is desirable to change the effective sampling rate in a sampled data system. In many cases, this can be accomplished simply by changing the sampling frequency to the ADC or DAC. However, it is often desirable to accomplish the sample rate conversion after the signal has been digitized. The most common techniques used are *decimation* (reducing the sampling rate by a factor of M), and *interpolation* (increasing the sampling rate by a factor of L). The decimation and interpolation factors (M and L) are normally integer numbers. In a generalized sample-rate converter, it may be desirable to change the sampling frequency by a non-integer number. In the case of converting the CD sampling frequency of 44.1kHz to the digital audio tape (DAT) sampling rate of 48kHz, interpolating by L =160 followed by decimation by M = 147 accomplishes the desired result.

The concept of decimation is illustrated in Figure 6.39. The top diagram shows the original signal, $f_a$, which is sampled at a frequency $f_s$. The corresponding frequency spectrum shows that the sampling frequency is much higher than required to preserve information contained in $f_a$, i.e., $f_a$ is oversampled. Notice that there is no information contained between the frequencies $f_a$ and $f_s - f_a$. The bottom diagram shows the same signal where the sampling frequency has been reduced (decimated) by a factor of M. Notice that even though the sampling rate has been reduced, there is no aliasing and loss of information. Decimation by a larger factor than shown in Figure 6.39 will cause aliasing.

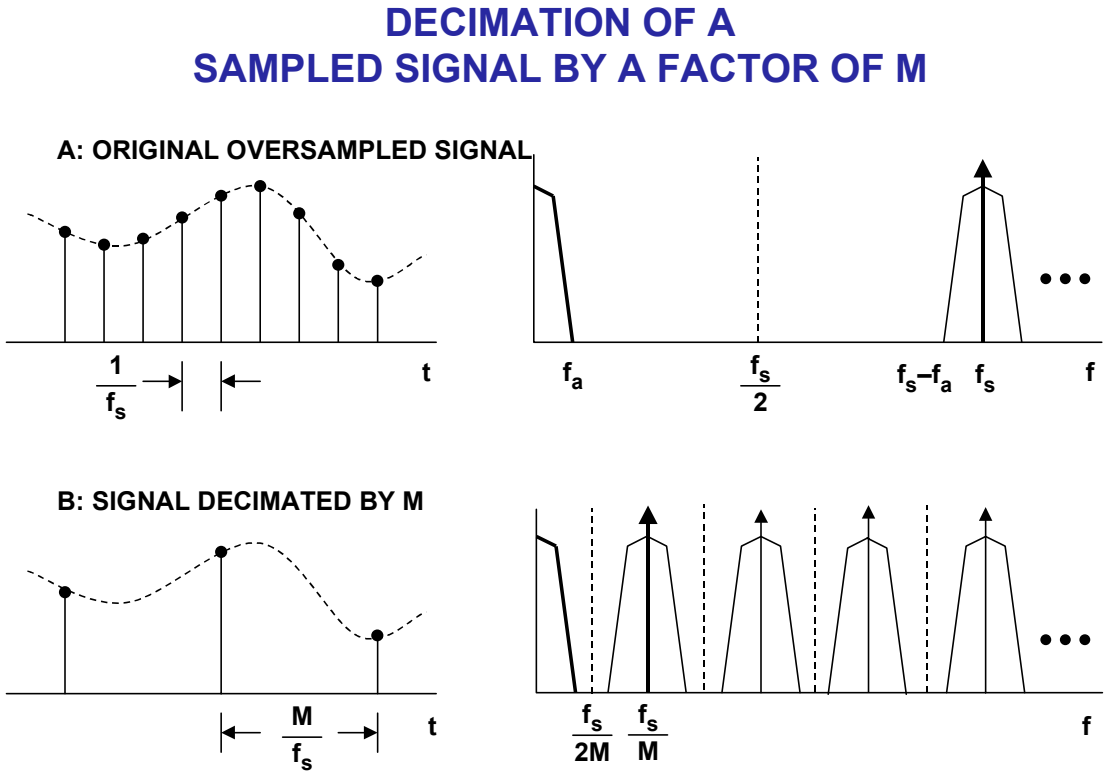



**Figure 6.39**

Figure 6.40A shows how to decimate the output of an FIR filter. The filtered data y(n) is stored in a data register which is clocked at the decimated frequency $f_S/M$. This does not change the number of computations required of the digital filter, i.e., it still must calculate each output sample y(n).

## DECIMATION COMBINED WITH FIR FILTERING

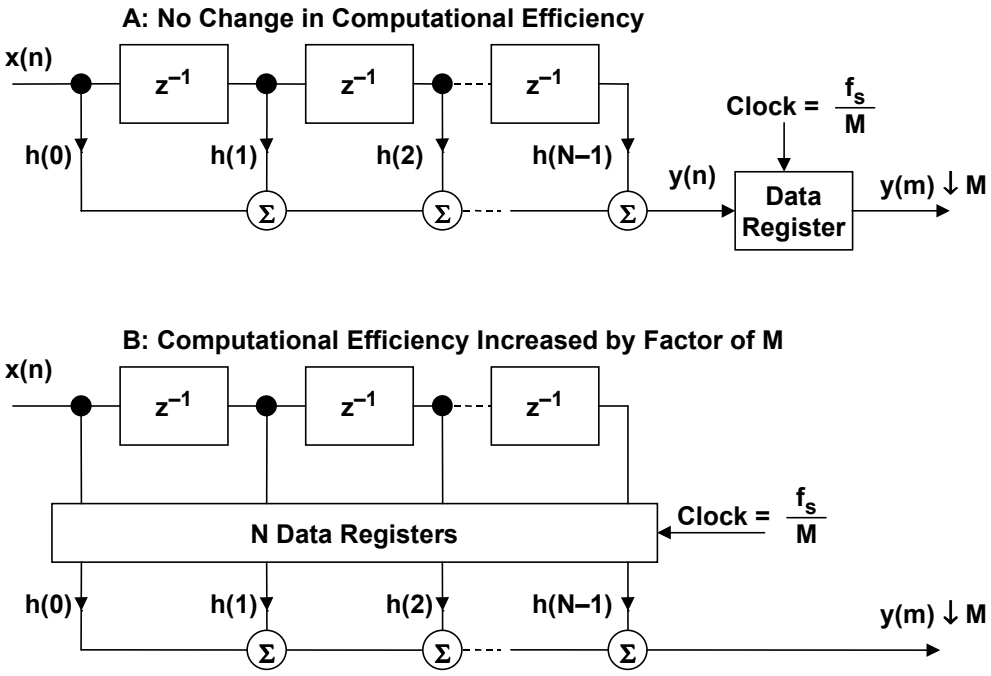



**Figure 6.40**

Figure 6.40B shows a method for increasing the computational efficiency of the FIR filter by a factor of M. The data from the delay registers are simply stored in N data registers which are clocked at the decimated frequency $f_S/M$. The FIR multiply/accumulates now only have to be done every Mth clock cycle. This increase in efficiency could be utilized by adding more taps to the FIR filter, doing other computations in the extra time, or using a slower DSP.

Figure 6.41 shows the concept of interpolation. The original signal in 6.41A is sampled at a frequency $f_S$. In 6.41B, the sampling frequency has been increased by a factor of L, and zeros have been added to fill in the extra samples. The signal with added zeros is passed through an interpolation filter which provides the extra data values.

## INTERPOLATION BY A FACTOR OF L

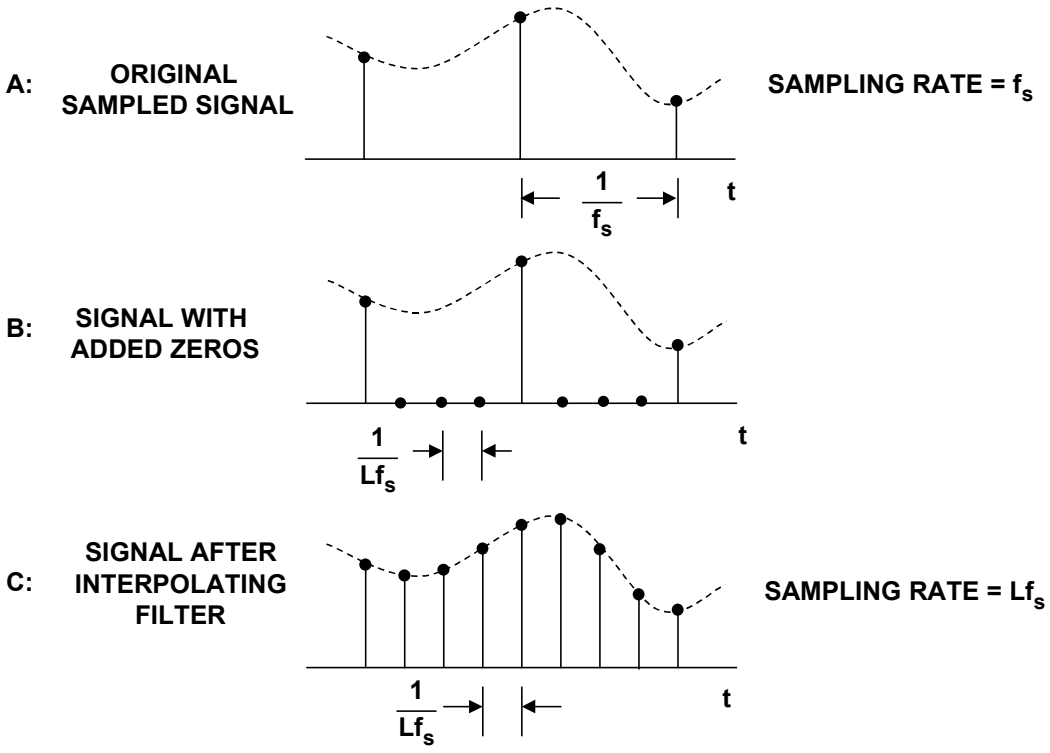



**Figure 6.41**

The frequency domain effects of interpolation are shown in Figure 6.42. The original signal is sampled at a frequency $f_S$ and is shown in 6.42A. The interpolated signal in 6.42B is sampled at a frequency $Lf_S$. An example of interpolation is a CD player DAC, where the CD data is generated at a frequency of 44.1kHz. If this data is passed directly to a DAC, the frequency spectrum shown in Figure 6.42A results, and the requirements on the anti-imaging filter which precedes the DAC are extremely stringent to overcome this. An oversampling interpolating DAC is normally used, and the spectrum shown in Figure 6.42B results. Notice that the requirements on the analog anti-imaging filter are now easier to realize. This is important in maintaining relatively linear phase and also reducing the cost of the filter.

The digital implementation of interpolation is shown in Figure 6.43. The original signal x(n) is first passed through a rate expander which increases the sampling frequency by a factor of L and inserts the extra zeros. The data then passes through an interpolation filter which smoothes the data and interpolates between the original data points. The efficiency of this filter can be improved by using a filter algorithm which takes advantage of the fact that the zero-value input samples do not require multiply-accumulates. Using a DSP which allows circular buffering and zero-overhead looping also improves efficiency.

## EFFECTS OF INTERPOLATION ON FREQUENCY SPECTRUM

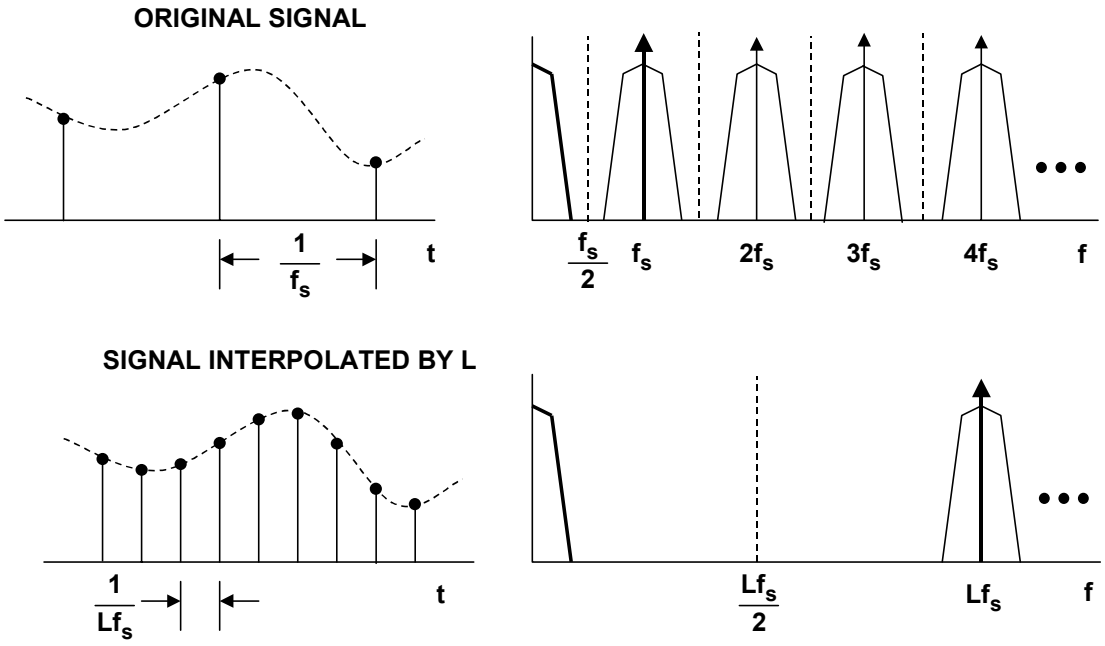
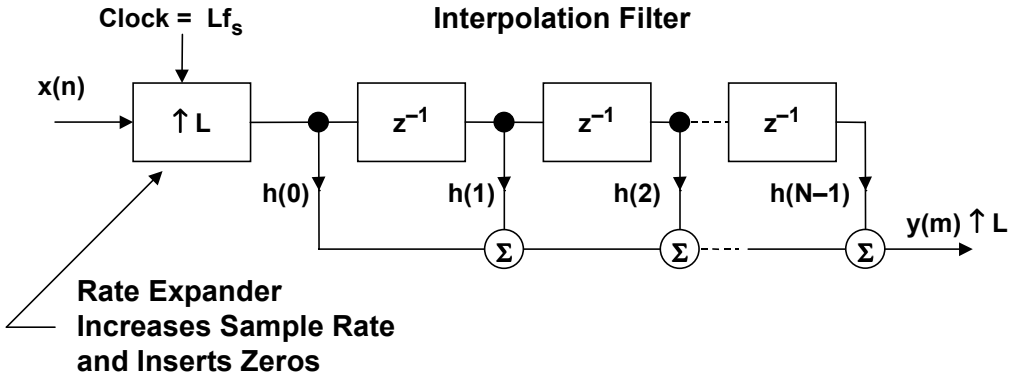



Figure 6.42

## TYPICAL INTERPOLATION IMPLEMENTATION




**Efficient DSP algorithms take advantage of:**

- Multiplications by zero
- Circular Buffers
- Zero-Overhead Looping

Figure 6.43

Interpolators and decimators can be combined to perform fractional sample rate conversion as shown in Figure 6.44. The input signal x(n) is first interpolated by a factor of L and then decimated by a factor of M. The resulting output sample rate is $Lf_S/M$. To maintain the maximum possible bandwidth in the intermediate signal, the interpolation must come before the decimation; otherwise, some of the desired frequency content in the original signal would be filtered out by the decimator.

An example is converting from the CD sampling rate of 44.1kHz to the digital audio tape (DAT) sampling rate of 48.0kHz. The interpolation factor is 160, and the decimation factor of 147. In practice, the interpolating filter h'(k) and the decimating filter h"(k) are combined into a single filter, h(k).

The entire sample-rate conversion function is integrated into the AD1890, AD1891, AD1892, and AD1893-family which operates at frequencies between 8kHz and 56kHz (48kHz for the AD1892). The new AD1896 operates up to 196kHz.

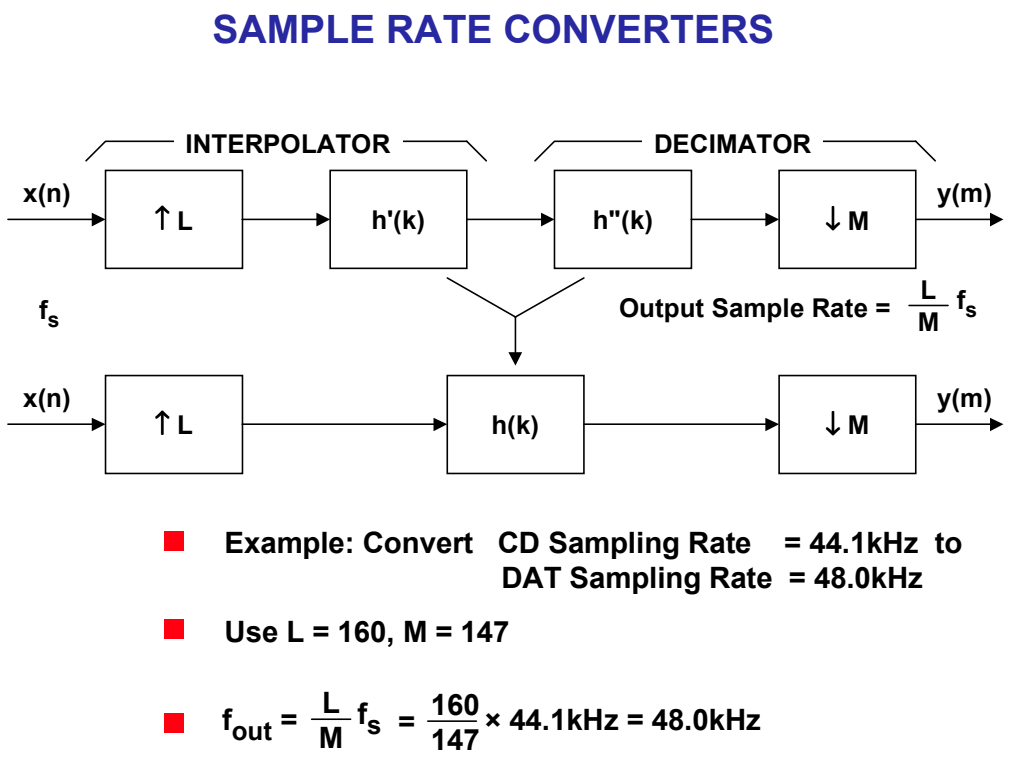



**Figure 6.44**

## ADAPTIVE FILTERS

Unlike analog filters, the characteristics of digital filters can easily be changed simply by modifying the filter coefficients. This makes digital filters attractive in communications applications such as adaptive equalization, echo cancellation, noise reduction, speech analysis and synthesis, etc. The basic concept of an adaptive filter is shown in Figure 6.45. The objective is to filter the input signal, x(n), with an adaptive filter in such a manner that it matches the desired signal, d(n). The

desired signal, d(n), is subtracted from the filtered signal, y(n), to generate an error signal. The error signal drives an adaptive algorithm which generates the filter coefficients in a manner which minimizes the error signal. The least-mean-square (LMS) or recursive-least-squares (RLS) algorithms are two of the most popular.

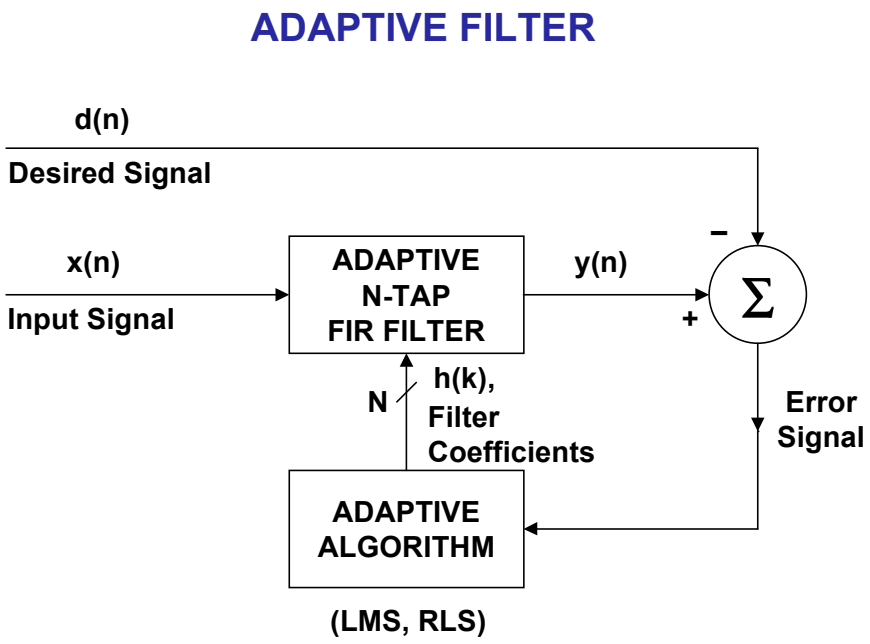



**Figure 6.45**

Adaptive filters are widely used in communications to perform such functions as equalization, echo cancellation, noise cancellation, and speech compression. Figure 6.46 shows an application of an adaptive filter used to compensate for the effects of amplitude and phase distortion in the transmission channel. The filter coefficients are determined during a training sequence where a known data pattern is transmitted. The adaptive algorithm adjusts the filter coefficients to force the receive data to match the training sequence data. In a modem application, the training sequence occurs after the initial connection is made. After the training sequence is completed, the switches are put in the other position, and the actual data is transmitted. During this time, the error signal is generated by subtracting the input from the output of the adaptive filter.

Speech compression and synthesis also makes extensive use of adaptive filtering to reduce data rates. The linear predictive coding (LPC) model shown in Figure 6.47 models the vocal tract as a variable frequency impulse generator (for voiced portions of speech) and a random noise generator (for unvoiced portions of speech such as consonant sounds). These these two generators drive a digital filter which in turn generates the actual voice signal.

## DIGITAL TRANSMISSION USING ADAPTIVE EQUALIZATION

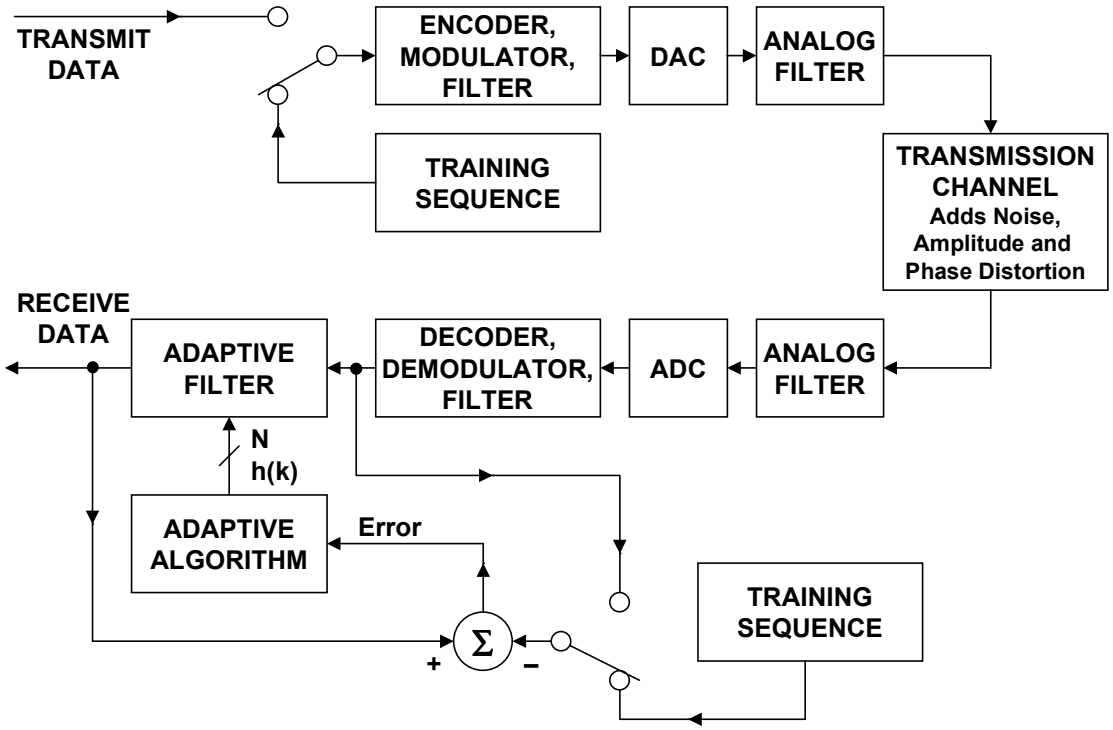



Figure 6.46

## LINEAR PREDICTIVE CODING (LPC) MODEL OF SPEECH PRODUCTION

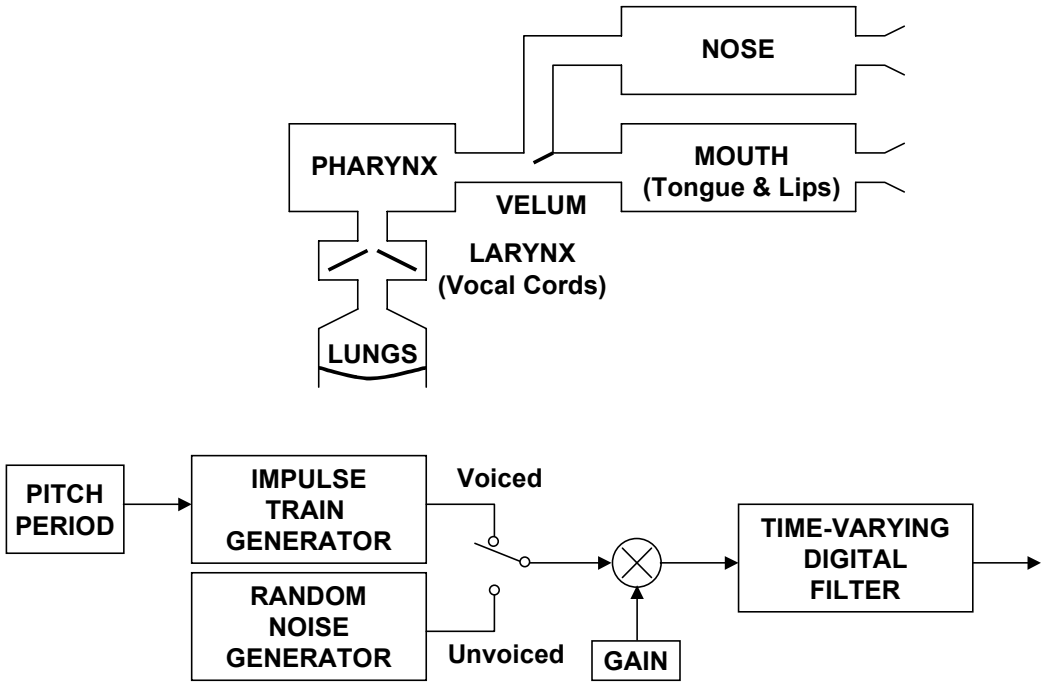



Figure 6.47

The application of LPC in a communication system such as GSM is shown in Figure 6.48. The speech input is first digitized by a 16-bit ADC at a sampling frequency of 8kSPS. This produces output data at 128kBPS which is much too high to be transmitted directly. The transmitting DSP uses the LPC algorithm to break the speech signal into digital filter coefficients and pitch. This is done in 20ms windows which has been found to be optimum for most speech applications. The actual transmitted data is only 2.4kBPS which represents a 53.3 compression factor. The receiving DSP uses the LPC model to reconstruct the speech from the coefficients and the excitation data. The final output data rate of 128kBPS then drives a 16-bit DAC for final reconstruction of the speech data.

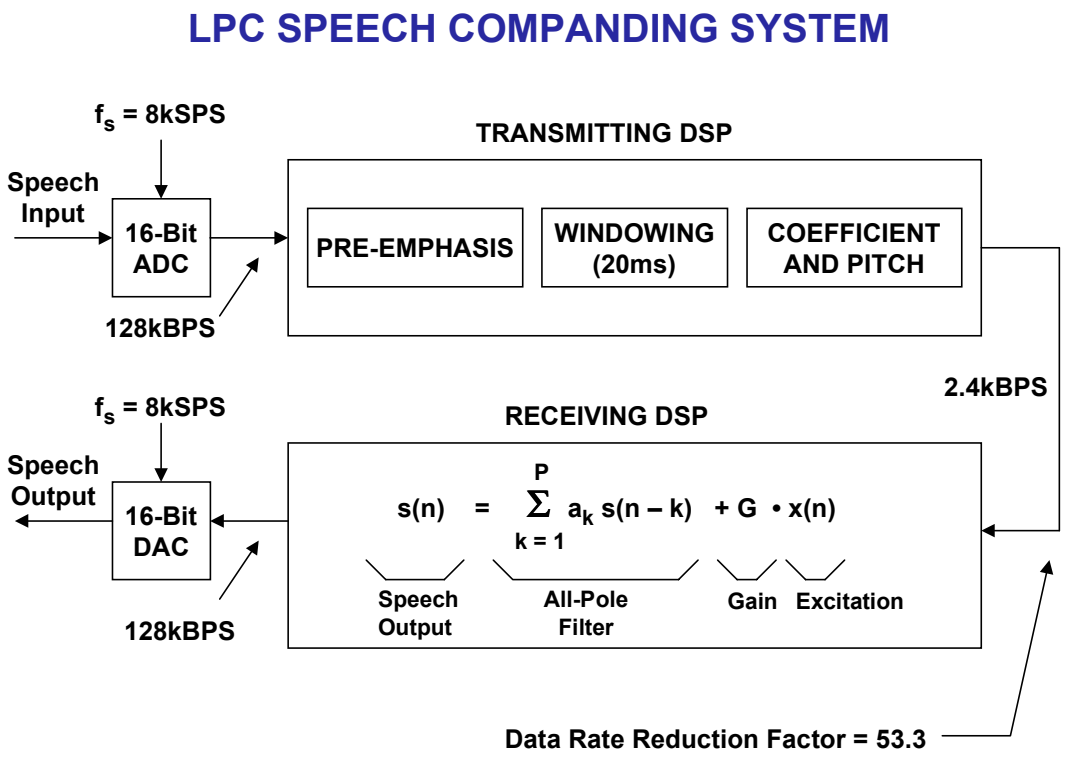



**Figure 6.48**

The digital filters used in LPC speech applications can either be FIR or IIR, although all-pole IIR filters are the most widely used. Both FIR and IIR filters can be implemented in a lattice structure as shown in Figure 6.49 for a recursive all-pole filter. This structure can be derived from the IIR structure, but the advantage of the lattice filter is that the coefficients are more directly related to the outputs of algorithms which use the vocal tract model shown in Figure 6.47 than the coefficients of the equivalent IIR filter.

The parameters of the all-pole lattice filter model are determined from the speech samples by means of linear prediction as shown in Figure 6.50. Due to the non-stationary nature of speech signals, this model is applied to short segments (typically 20ms) of the speech signal. A new set of parameters is usually determined for each time segment unless there are sharp discontinuities, in which case the data may be smoothed between segments.

## ALL POLE LATTICE FILTER

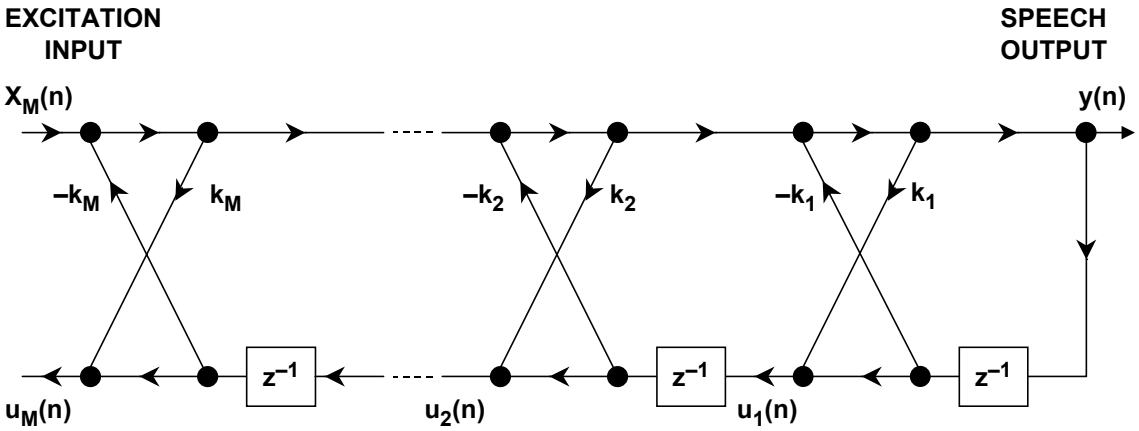



Figure 6.49

## ESTIMATION OF LATTICE FILTER COEFFICIENTS IN TRANSMITTING DSP

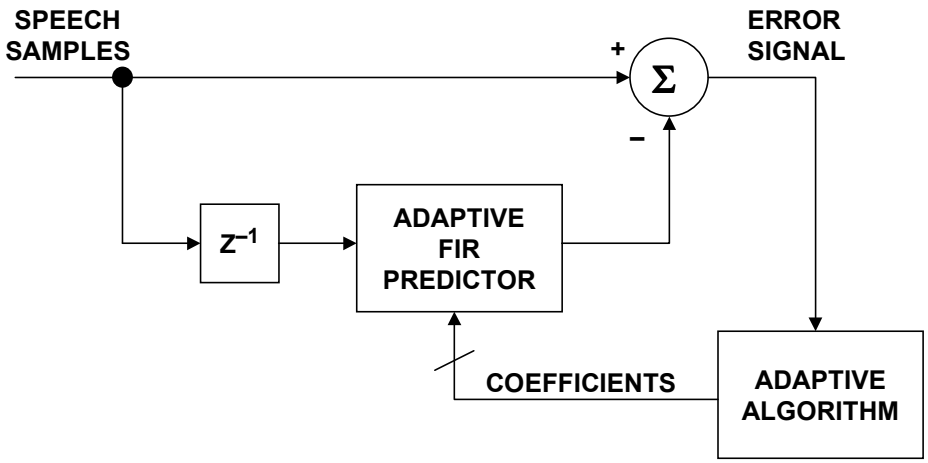



Figure 6.50

# REFERENCES


1. Steven W. Smith, **The Scientist and Engineer's Guide to Digital Signal Processing**, Second Edition, 1999, California Technical Publishing, P.O. Box 502407, San Diego, CA 92150. Also available for free download at: http://www.dspguide.com or http://www.analog.com

2. C. Britton Rorabaugh, **DSP Primer**, McGraw-Hill, 1999.

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

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

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

6. John G. Proakis and Dimitris G. Manolakis, **Introduction to Digital Signal Processing**, MacMillian, 1988.

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

8. 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.

9. Momentum Data Systems, Inc., 17330 Brookhurst St., Suite 140, Fountain Valley, CA 92708, http://www.mds.com

10. **Digital Signal Processing Applications Using the ADSP-2100 Family**, Vol. 1 and Vol. 2, Analog Devices, Free Download at: http://www.analog.com

11. **ADSP-21000 Family Application Handbook**, Vol. 1, Analog Devices, Free Download at: http://www.analog.com

12. B. Widrow and S.D. Stearns, **Adaptive Signal Processi**ng, Prentice-Hall, 1985.

13. S. Haykin, **Adaptive Filter Theory**, 3rd Edition, Prentice-Hall, 1996.

14. Michael L. Honig and David G. Messerschmitt**, Adaptive Filters - Structures, Algorithms, and Applications**, Kluwer Academic Publishers, Hingham, MA 1984.

15. J.D. Markel and A.H. Gray, Jr., **Linear Prediction of Speech**, Springer-Verlag, New York, NY, 1976.

16. L.R. Rabiner and R.W. Schafer**, Digital Processing of Speech Signals**, Prentice-Hall, 1978.