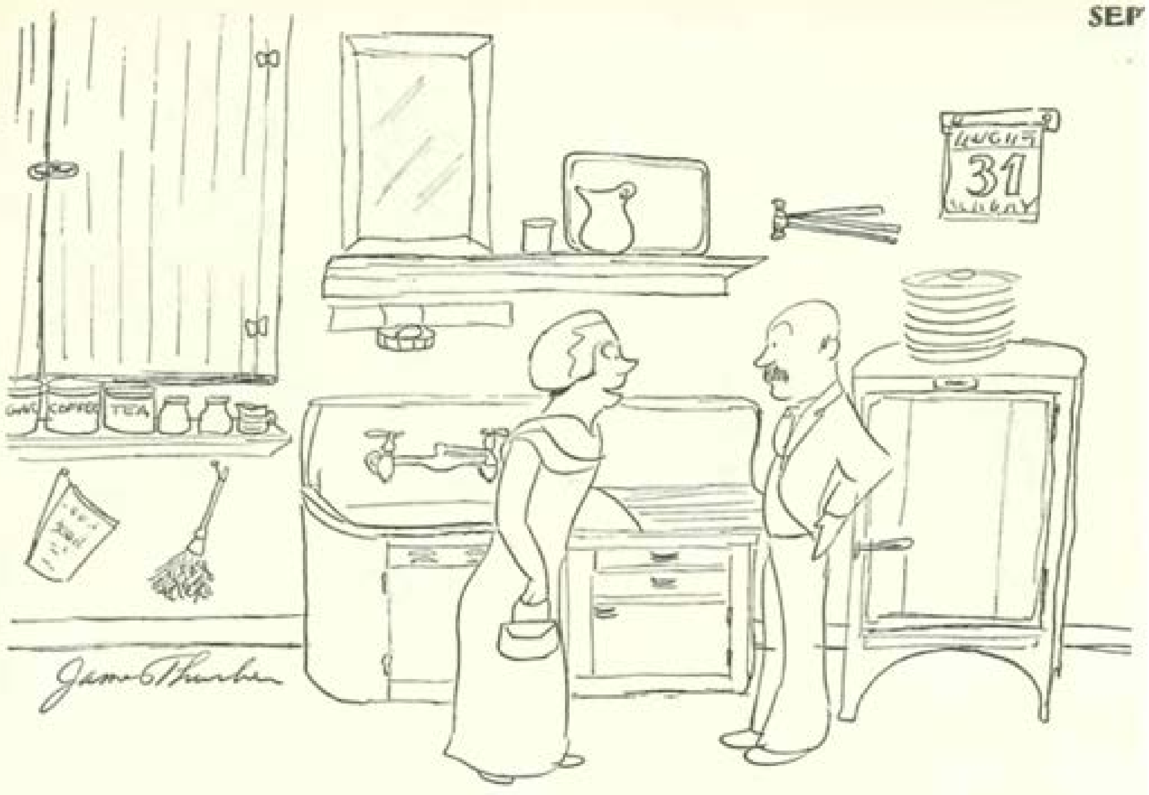


"Are there any—cucarachas?"

EE 311 **April 19, 2019**
Lecture 24

7.3 Conversion by a Rational Factor

There are two ways to put together an interpolator and a decimator: we can either up-sample by *U* and down-sample by *D* or vice-versa. If *U* and *D* have no common factors both methods will result in the same signal. However, if we down-sample by *D* first we introduce image frequencies at the reduced sample rate. If, on the other hand, we up-sample by *U* first we introduce image frequencies at the multiplied sample rate and since these are further in frequency space from the signal they are easier to deal with. Putting the up-sampler first makes good intuitive sense as well since the interpolation filter does not have to fill in such wide gaps between samples. Figure 7.13 shows this configuration. Note that we can put the anti-aliasing filter and the anti-imaging filter together in a single filter which meets the more severe frequency specifications.

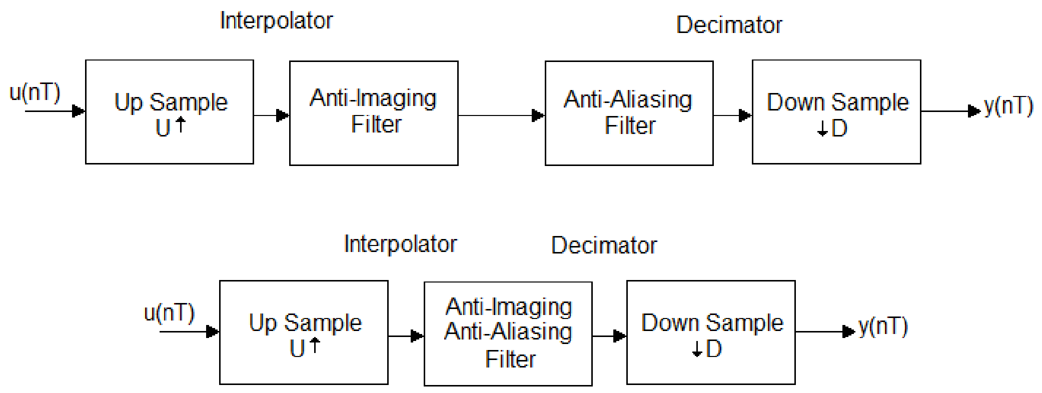


Figure 7.13
Combining an interpolator and a decimator to do rational number frequency conversion.

Example 7.6
Given a signal which is band limited to 100 Hz and is sampled at 1000 Hz, design a rational rate converter to change the sample rate to 2,500 Hz.

Solution
The common factor between 2,500 and 1,000 is 500 which gives us an up/down ratio of 5/2. We need a factor of 5 interpolator and a factor of 2 decimator.

Anti-imaging filter
The anti-imaging filter will run at 5 x 1000 = 5 KHz. The original signal is sampled at 1 KHz and is limited to the first 100 Hz so the first image will be at 1000 – 100 = 900 Hz. The pass band will be from 0 to 100 Hz and the stop band will run from 900 Hz to 2.5 KHz.

Anti-aliasing filter
The anti-aliasing filter will be running at 5 KHz. If we down-sample by 2 we will introduce an alias from 2.4 KHz to 2.5 KHz. The pass band will be from 0 to 100 Hz and the stop band will be from 2.4 KHz to 2.5 KHz.

The anti-imaging filter has the more stringent requirements so the anti-aliasing filter need not be implemented. If we take the pass band and stop band ripple to be 0.001 we can implement the anti-imaging filter with a Parks-McClelland filter of order 19. Figure 7.14 shows the rate converter along with signals in the time and frequency domains.

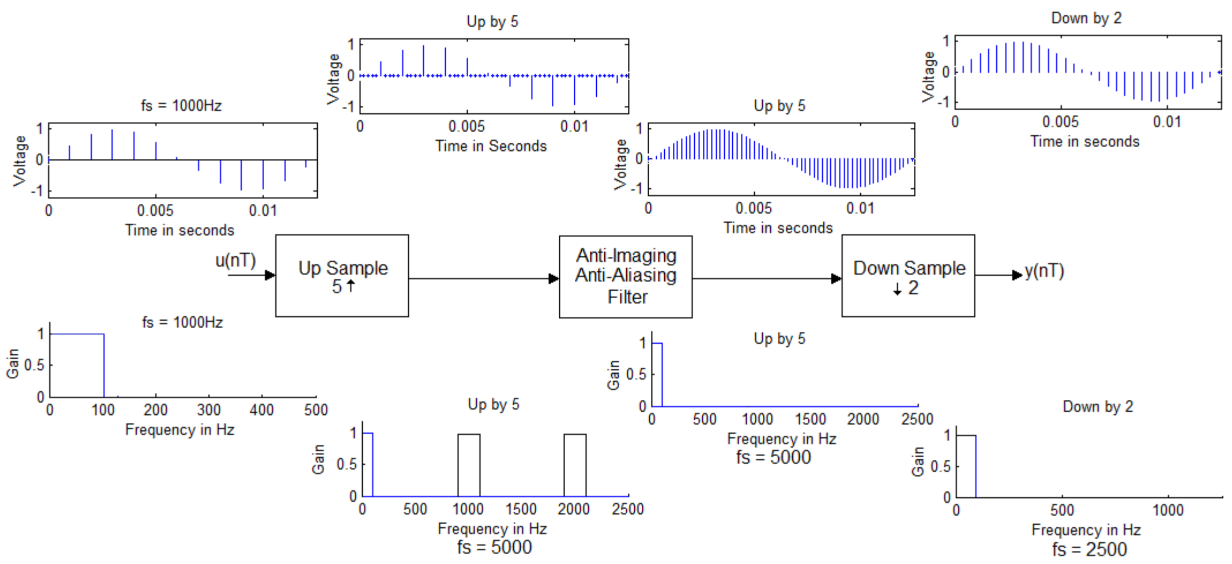


Figure 7.14
A rate conversion system that changes the sample frequency from 1000 Hz to 2500 Hz using a cascaded interpolator and decimator is shown.   Figures in the top row are in the time domain; figures in the bottom row are in the frequency domain.

7.4 FIR Implementation

A decimator consists of a low pass filter and a down-sampler which reduces the sample rate by a factor of *D*. The down-sampler outputs only one in every *D* samples and ignores the remainder. Thus, some efficiency can be gained in the low pass filter by not computing those samples which are not going to be used. Likewise, an interpolator uses an up-sampler which inserts *U* zeros between samples and passes the result to an interpolation filter. The interpolation filter need not do the multiplications where there are zeros in the input so some significant computational advantage can be gained by taking advantage of the up-sampling. In this section we look at more efficient ways to implement the FIR filters used in decimators and interpolators.

Decimation filters

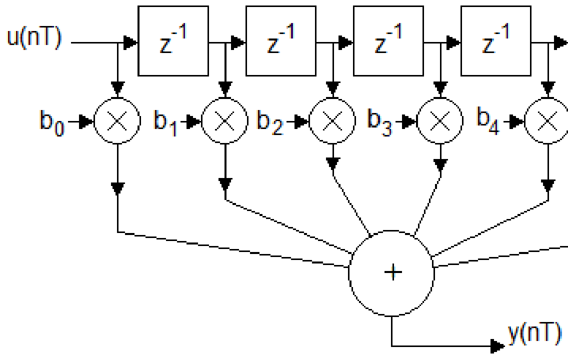


Figure 7.15
A standard FIR filter implementation for a filter of order 5.

The canonic structure of an FIR filter is shown in Figure 7.15. This structure implements the general difference equation given by

$$y[n] = b_0 u[n] + b_1 u[n-1] + \cdots + b_M u[n-M] = \sum_{k=0}^{M} b_k u[n-k]$$

Consider down-sampling by a factor D = 2 and an order 5 FIR filter. Table 7.2 shows how data would move through such a filter.

		Multiplying coefficients						FIR output	Down Sample
		b_0	b_1	b_2	b_3	b_4	b_5		
←Time		• • •							
	5	$u[5]$	$u[4]$	$u[3]$	$u[2]$	$u[1]$	$u[0]$	$y[5]$	--
	6	$u[6]$	$u[5]$	$u[4]$	$u[3]$	$u[2]$	$u[1]$	$y[6]$	$y_D[3]$
	7	$u[7]$	$u[6]$	$u[5]$	$u[4]$	$u[3]$	$u[2]$	$y[7]$	--
	8	$u[8]$	$u[7]$	$u[6]$	$u[5]$	$u[4]$	$u[3]$	$y[8]$	$y_D[4]$
	9	$u[9]$	$u[8]$	$u[7]$	$u[6]$	$u[5]$	$u[4]$	$y[9]$	--
	10	$u[10]$	$u[9]$	$u[8]$	$u[7]$	$u[6]$	$u[5]$	$y[10]$	$y_D[5]$
	11	$u[11]$	$u[10]$	$u[9]$	$u[8]$	$u[7]$	$u[6]$	$y[11]$	--
	12	$u[12]$	$u[11]$	$u[10]$	$u[9]$	$u[8]$	$u[7]$	$y[12]$	$y_D[6]$
		• • •							

Table 7.2
This table shows how data moving through an order 5 FIR filter. The right most column shows which data is used by the following down-sampler if the down-sample factor is 2. [4]

From the table we note that in the column labeled b_0 we are using only $u[6]$, $u[8]$, $u[10]$, … which is a down-sampled version of the input stream. Likewise, the column headed b_1 is a delayed version of column b_0 and we are using $u[5]$, $u[7]$, $u[9]$,… which is a down-sampled and delayed version of the input stream. This would suggest the structure shown in Figure 7. 15 for implementation.

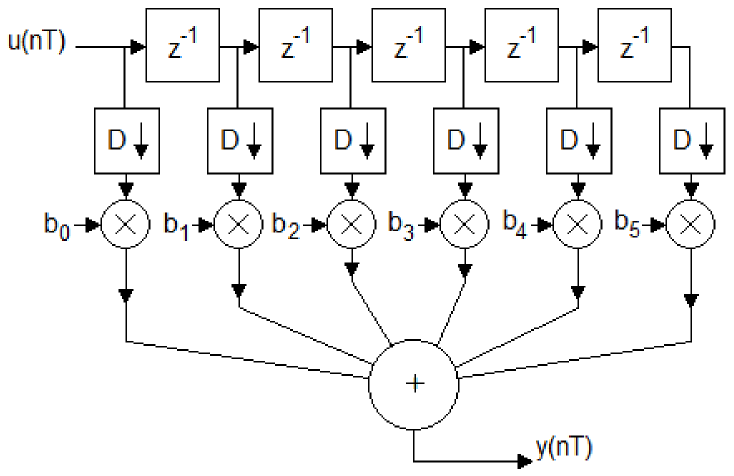


Figure 7.16
Putting the down-samplers prior to the multipliers eliminates those samples that do not contribute to the output.

In Figure 7.16 the multiplications take place after the down-sampling which means that they can run at a slower rate. This assumes that the down-samplers are storing their output in a buffer which implies additional storage space. In software, Figure 7.17 shows the c-code to implement this efficient algorithm and take advantage of the fact that the output need not be computed for every input sample. The processor specific code has been replaced with comments.

There is example code in the book – see for yourself.

Interpolation filters
Interpolation filters can be implemented as either FIR or IIR filters, but since IIR filters tend to be more computationally efficient, one would think that they would be the choice except in cases where linear phase was needed. This is not the case however, because the interpolation filter always follows an up-sampler which has inserted zeros into the data stream. For an IIR filter, these zeros get mixed with the non-zero data in the feedback of the filter and there is no good way to ignore them. For the FIR filter however, there is no feedback, the zeros are not intermixed with the non-zero data, and it is possible to construct an FIR architecture which effectively ignores the zeros. This leads to greater computational efficiency for two reasons: first, the zeros are ignored so no multiplication and no addition is needed for these terms. The second reason is that it becomes possible to implement FIR interpolation filters with some degree of parallelism.

To understand how this works, consider a specific simple example. Suppose we have an interpolation filter which follows an up-sampler which doubled the sample frequency by inserting a zero between every two samples. Further, suppose the filter has order 5, length 6 and the coefficients are b_0, b_1, b_2, b_3, b_4, and b_5 as shown in Figure 7.16. The data stream coming into the up-sampler is given by:
$u[n] = \{ u[0], u[1], u[2], \dots u[n], \dots\}$.
The up-sampler adds zeros between samples so the data stream coming into the filter is given by:
$u_{up}[n] = \{u[0], 0, u[1], 0, u[2], 0, u[3], 0, \dots u[n], 0, \dots\}$

Table 7.3 shows the multiplications as the data passes through the filter.

		Multiplying coefficients						**Non-zero multipliers**
		b_0	b_1	b_2	b_3	b_4	b_5	
Time ↓		…						
	4	$u[2]$	0	$u[1]$	0	$u[0]$	0	b_0, b_2, b_4
	5	0	$u[2]$	0	$u[1]$	0	$u[0]$	b_1, b_3, b_5
	6	$u[3]$	0	$u[2]$	0	$u[1]$	0	b_0, b_2, b_4
	7	0	$u[3]$	0	$u[2]$	0	$u[1]$	b_1, b_3, b_5
	8	$u[4]$	0	$u[3]$	0	$u[2]$	0	b_0, b_2, b_4
	9	0	$u[4]$	0	$u[3]$	0	$u[2]$	b_1, b_3, b_5
	10	$u[5]$	0	$u[4]$	0	$u[3]$	0	b_0, b_2, b_4
	11	0	$u[5]$	0	$u[4]$	0	$u[3]$	b_1, b_3, b_5
	12	$u[6]$	0	$u[5]$	0	$u[4]$	0	b_0, b_2, b_4
	13	0	$u[6]$	0	$u[5]$	0	$u[4]$	b_1, b_3, b_5
	14	$u[7]$	0	$u[6]$	0	$u[5]$	0	b_0, b_2, b_4
	15	0	$u[7]$	0	$u[6]$	0	$u[5]$	b_1, b_3, b_5
	16	$u[8]$	0	$u[7]$	0	$u[6]$	0	b_0, b_2, b_4
	17	0	$u[8]$	0	$u[7]$	0	$u[6]$	b_1, b_3, b_5
	18	$u[9]$	0	$u[8]$	0	$u[7]$	0	b_0, b_2, b_4
		…						

Table 7.3
This table shows how data with zeros between the sample points flows through the filter of Figure 7.15. The column at right shows which coefficients are used.

It is clear from the table that in even numbered time periods we are multiplying by coefficients b_0, b_2, and b_4 and in odd numbered time periods we are multiplying by coefficients b_1, b_3, and b_5. At no time do we need to simultaneously multiply by all six coefficients. This would suggest that we can think of this filter as consisting of two somewhat independent filters running in alternate time periods and that each filter is significantly smaller than the whole filter. Figure 7.18 shows this configuration. In this figure, both filters receive the same input but they use a different set of coefficients to calculate the output. A multiplexor selects which filter's output ultimately becomes the real output. The filters are independent except they share a common input and their output must be synchronized so they both run from the same clock. Each filter could be constructed from an independent computer system or possibly a microcontroller.

For the configuration shown the clock rate is at the up-sampled rate and the individual filter output must be calculated at this rate. Additional time could be gained by providing a buffer to hold the delayed input values between samples.

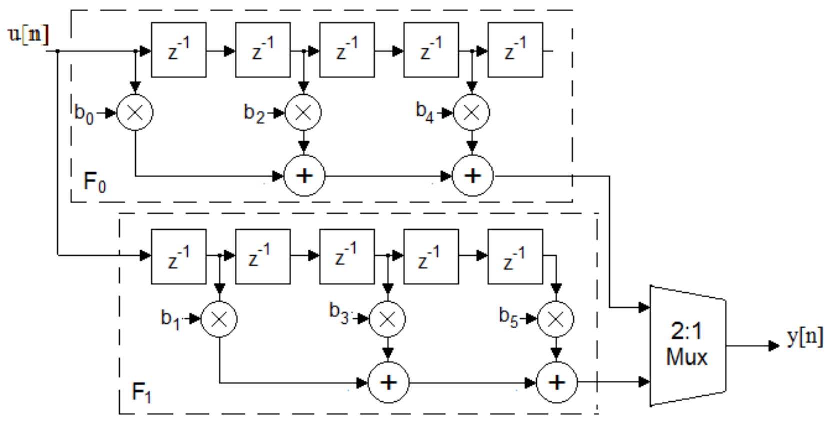


Figure 7.18
Efficient implementation schematic for an interpolation filter that follows a factor 2 up-sampler.

7.5 Narrow Band Filters
Skip

7.6 Conversion by an Arbitrary Factor
Skip

7.7 Bandpass Sampling
Bandpass sampling which is often called *under sampling* is another technique that can be used to alter the sample rate when the signal being considered is limited to some band of frequencies such as those signals created by modulation of a carrier frequency by a signal. In Figure 7.28 a low pass band of signal sampled at f_s will produce aliases at multiples of the sample frequency. Likewise, and counter to intuition, if we have a bandpass signal with upper and lower frequencies f_U and f_L and center frequency $f_c = nf_s$ we will get the identical frequency spectrum if we sample it at f_s.

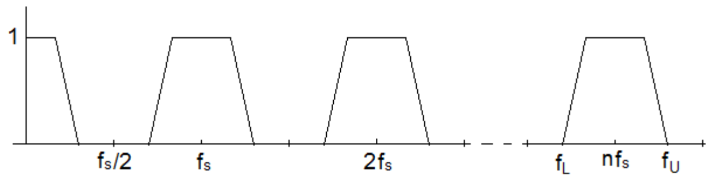


Figure 7.28
Aliases of a low pass band.

This concept can be used to recover signal information from a modulated carrier without having to sample it at twice its highest frequency, or $2f_U$. Example 7.8 illustrates how this works in MATLAB®.

Example 7.9
Consider the single sideband analog signal shown in Figure 7.29 where we will take f_L = 10 KHz and f_U = 10.2 KHz. We want to sample this bandpass signal at frequency f_s such that we move the band to a starting frequency of 0 Hz. The bandwidth is 200 Hz so it is clear that our sample

frequency will have be greater than 400 Hz to recover the signal in the band. We also want the frequency f_L to be moved to 0 Hz so that f_L must be an integer multiple of f_s. If we take f_s = 1 KHz we satisfy both of these conditions.

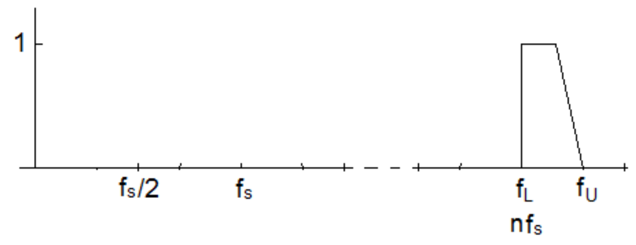


Figure 7.29
An analog bandpass signal.

In MATLAB® we can represent the band of signals between f_L and f_U as a sequence of sinusoids at 10 KHz, 10.05 KHz, 10.1 KHz, 10.15 KHz, and 10.2 KHz and sample each at 1 KHz like this:

```
fs = 1000;T = 1/fs;
BW = 200;
fL = 10000;
fU = fL + BW;
t = 0:T:.1;
ud = zeros(1, length(t));
for fr = fL:50:fU
    ud = ud + sin(2*pi*fr*t);
end
```

We then use the fast Fourier Transform (fft) to find the frequency spectrum of the bandpass signal which has been sampled at 1 KHz.

```
fftud = fft(ud)/length(ud);
k = 0:length(t)-1;
stem(k*fs/(length(t)-1), abs(ftud));
axis([0 fs1/2 0 1]);
```

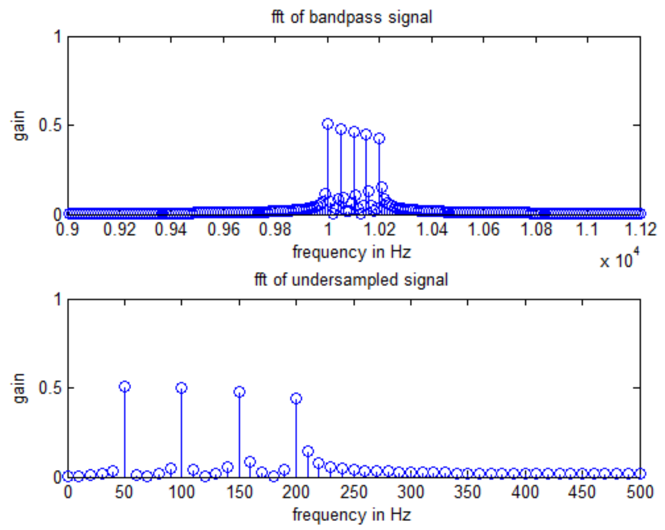


Figure 7.30
The top figure shows the original signal in the range of 10 KHz to 10.2 KHz. The bottom figure shows the original signal after it has been sampled at 1 KHz.

For band pass sampling the sample frequency needs to be at least twice the bandwidth of the signal. In addition, if we want the lower band pass edge to map to frequency 0 Hz, it is necessary that the lower edge be a multiple of the sample frequency.

Band pass sampling is not without its problems. Note that the under sampled signal not only captures the signal in the pass band, it also captures the "empty" bands which are multiples of the pass band. These typically contain noise so that the resulting signal has the noise from multiple bands folded into it. The signal to noise ratio is typically much worse.

In the general case the lower edge of the pass band is not a multiple of the sample frequency as shown in Figure 7.31. For this figure we can write

$\frac{(n-1)f_s}{2} \le f_L$ which can be written as

$$\frac{f_s}{2} \le \frac{f_L}{n-1} \qquad 7.1$$

Likewise, at the upper end we get

$f_H \le \frac{nf_s}{2}$ which can be written

$$\frac{f_s}{2} \ge \frac{f_H}{n} \qquad 7.2$$

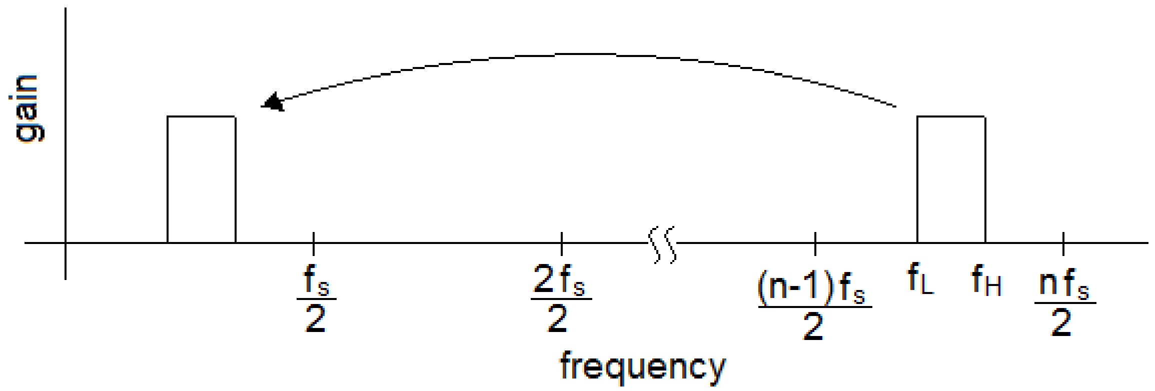


Figure 7.31
The general case of bandpass sampling. The lower frequency edge f_L is not a multiple of f_s.

Putting equations 7.1 and 7.2 together we get the expression for the *bandpass sampling theorem.*

$$\frac{2f_H}{n} \le f_s \le \frac{2f_L}{n-1}$$

Where n = 2, 3, … int(f_H/bandwidth)

7.8 Oversampling in Audio Applications

The range of the human ear is from about 20 Hz to 20 KHz and digital audio systems have standardized on 44.1 KHz as the sampling rate. To accurately record high quality digital audio we need a "brick wall" analog anti-aliasing filter on the front end of an analog to digital converter. Any noise introduced by other components in the recording process that has a frequency above 22,050 Hz will be aliased back into the range of interest unless it is first eliminated by such a filter. True brick wall filters with a cutoff frequency of 22,050Hz can only be approximated and can be very expensive. In addition, analog components suffer from the effects of temperature, humidity, and aging and can be difficult to maintain. Oversampling provides a clever way to replace the brick wall anti-aliasing filter with one that meets much less stringent requirements. When over sampling is combined with an A/D converter the system is referred to as an *oversampled A/D converter*.

To better understand how this works consider a system that uses oversampling by a factor of 2 with an A/D converter. We want the digital signal coming out of this system to be at f_s = 44.1 KHz so the over sampled system is running at $2f_s$ = 88.2 KHz sample rate. With the sample rate up by a factor of 2, the pass band will range from 0 to $f_s/2$ but the stop band will range from $1.5f_s$ to infinity as shown in Figure 7.32. With a ripple in the pass and stop bands of 0.01 such specifications could be met with a 6th order Butterworth filter.

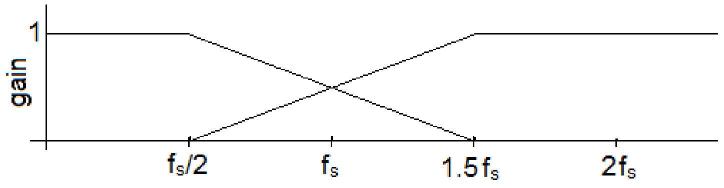


Figure 7.32
Analog filter pass band and stop band. If the analog signal passing through this filter is sampled a $2f_s$, the frequencies below $f_s/2$ will be preserved without aliasing.

The signal coming from the anti-aliasing filter will be passed through a sample and hold at 2fs and converted to a digital sequence. The A/D converter can be followed by a low pass digital filter and down-sampled to 44.1 KHz. Figure 7.33 shows the whole process.

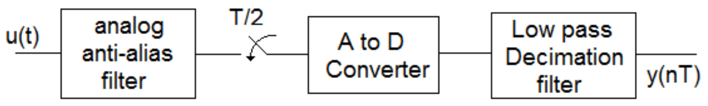


Figure 7.33
An over sampling A/D converter. The signal is up-sampled by a factor of 2, converted to a digital sequence, and passed through a decimation filter which reduces the sampling rate to f_s and blocks frequencies above $f_s/2$.

The decimation filter on the output not only reduces the sample frequency to the f_s, it also serves as a more stringent anti-aliasing filter eliminating those frequencies above $f_s/2$. Up-sampling by a higher factor further reduces requirements on the analog filter but requires an A/D converter that runs at a higher frequency.

On the output of an audio system we have a similar problem with the anti-imaging filter that follows a D/A converter. In this case, we can up-sample before passing the signal to the D/A converter thereby reducing the specifications on the analog anti-imaging filter and allowing for less expensive analog components.

The ideal anti-imaging filter is a low pass filter with a gain of one from 0 to $f_s/2$ and a gain of zero thereafter. Up-sampling before passing the signal to the D/A converter allows us to use a digital low pass filter to reduce the specifications for the analog anti-imaging filter. Figure 7.34 shows how this is done. In this figure, the incoming signal is sampled at f_s. Up-sampling followed by an interpolation filter allows us to move the alias image to a higher frequency range. The result is the specifications on the analog anti-imaging filter are less stringent.

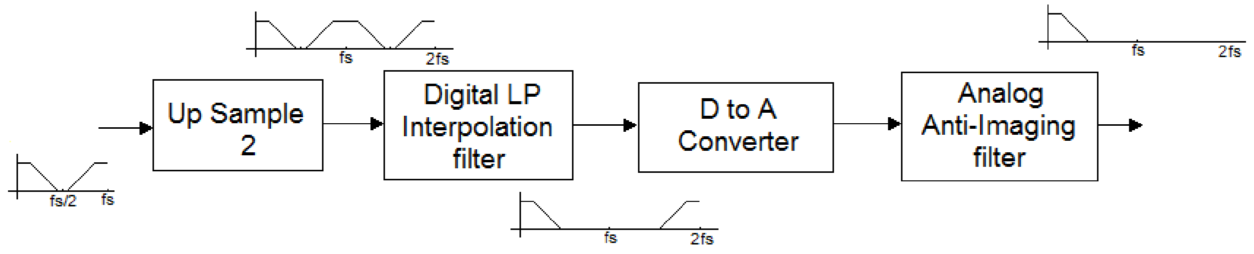


Figure 7.34
Up-sampling reduces the requirements on the analog anti-imaging filter.

Chapter 8
Realization and Implementation of Digital Filters

8.1 Implementation Issues

The error introduced into the implementation process may be categorized as follows:

1. Input quantization error. The input to a digital filter is typically a continuous time signal which is sampled and passed through an A/D converter. The A/D changes the continuous time signal to a binary number which has finite length. Thus the input is quantized and only approximates the continuous time sample.

2. Aliasing error. By their nature, continuous time signals are not band limited and any input frequency above the Nyquist rate will be aliased to a frequency below the Nyquist rate.

3. Coefficient quantization error. Just as the input is quantized, the filter coefficients are also quantized since we use a fixed number of bits representing each coefficient. Thus, the transfer function which is implemented is only an approximation to the transfer function which was designed.

4. Product quantization error. Multiplication of two *n*-bit numbers results in a product which has 2*n* bits. Products must be adjusted to the number of bits that fit in the registers resulting in error.

5. Overflow error. Overflow error results when a number is too large to represent in the number of bits being used in the system. Overflow can take place at the input to the A/D, or it can happen during the course of calculations.

6. Reconstruction error. This is the error that is introduced when we attempt to reconstruct a continuous time continuous amplitude signal from the output of a discrete system.

8.2 Number Representation
Skip

8.3 Realization Structures

A causal DSP system can be represented by a transfer function in *z* which can be rendered in the time domain as a difference equation. To implement the difference equation we need only three unique operations: multiplication by a constant, addition, and time delay. We can use these three operations to represent a transfer function as a structure. Manipulation of the structure is equivalent to manipulation of the difference equation with the advantage of a visual presentation which shows the relationships between the variables. The structure is similar in form to a signal flow graph and some authors use signal flow graphs for realization structures. We will use the three symbols in a block diagram format as shown in the following example.

Example 8.1
Represent the transfer function given by

$$H(z) = \frac{z+1}{z^2 - .9}$$

as a block diagram using multiplication by a constant, unit delay, and addition blocks.

Solution

The *z* transform of the difference equation corresponding to the transfer function is given by $y(z) = z^{-1}x(z) + z^{-2}x(z) + .9z^{-2}y(z)$. One version of a block diagram that serves as a realization for this difference equation is given in Figure 8.5.

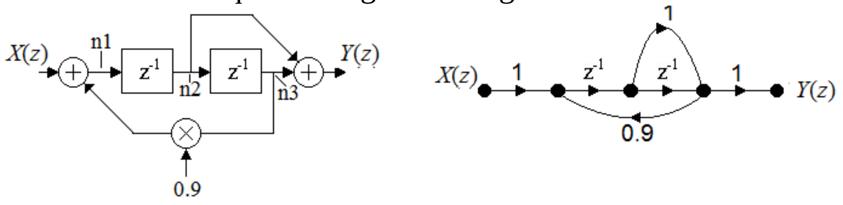


Figure 8.5
(Left) Block diagram representing a difference equation. (Right) The equivalent signal flow graph.

To verify this we can write the following four equations:

$$n1 = X(z) + 0.9n3$$

$$n2 = z^{-1}n1$$

$$n3 = z^{-1}n2$$

$$Y(z) = n2 + n3$$

From these four equations we can eliminate the node variables *n*1, *n*2, and *n*3 to get

$$Y(z) = z^{-1}X(z) + z^{-2}X(z) + .9z^{-2}Y(z)$$

$$Y(z) = \frac{z^{-1} + z^{-2}}{1 - 0.9z^{-2}} X(z) = \frac{z+1}{z^2 - 0.9} X(z)$$

FIR Structures

It might at first seem surprising that the difference equation for an FIR filter can be rewritten in many different realizations. Here we consider six of them: direct, transposed, symmetrical, cascaded, parallel, and lattice. Each has its own particular set of advantages and disadvantages and choosing the proper realization for implementation is part of the design process.

Direct Realization

We think of an FIR filter as little more than a tapped delay line as shown in Figure 8.6. We will refer to this configuration as the *direct realization* for an FIR filter.

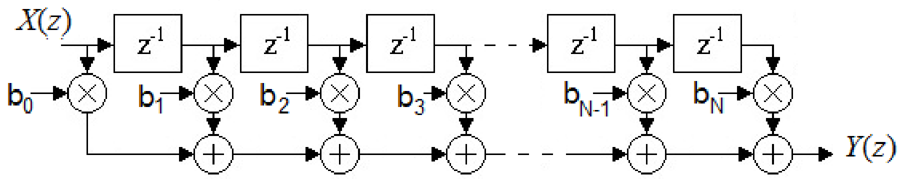


Figure 8.6
Direct form realization for an *N*th order FIR filter. The impulse response is $h[n] = \{b_0, b_1, \cdots b_N\}$.

Transposed Realization
A different version of the direct form can be created using the *flow graph transposition theorem.* According to this theorem, a transposed system can be obtained by reversing all of the branches and swapping the input with the output. For a LTI system with a single input and a single output, the transposed system has the same transfer function as the original system. Figure 8.7 shows the transposed version of the direct form FIR filter in Figure 8.6.

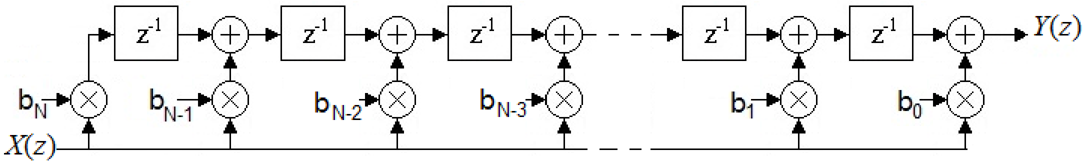


Figure 8.7
Transposed direct form for an Nth order FIR filter.

The direct form and the transposed direct form both require the same number of multiplications, additions, and storage locations. The transposed form has a slight disadvantage in that the number being shifted through the delay line is a product. If guard bits are used to preserve product precision, the storage required could be larger.

Symmetrical Realization
For FIR filters which have linear phase, the coefficients are either symmetrical or anti-symmetrical. The symmetrical form shown in Figure 8.8 takes advantage of this symmetry to reduce the number of multiplications.

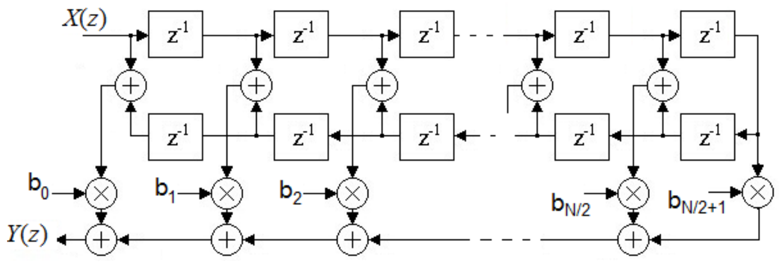


Figure 8.8
A symmetrical structure for FIR filters. In this case $b_0 = b_N$, $b_1 = b_{N-1}$, etc.

In Figure 8.8, the order is odd and the length is even so that the difference equation can be written as:

$$\begin{aligned} y[n] &= b_0(x[n]+x[n-N-1]) + b_1(x[n-1]+x[n-N]) + \cdots \\ &+ b_{N/2}(x[n-N/2]+x[n-N/2+2]) + b_{N/2+1}x[n-N/2+1] \end{aligned}$$

The clear advantage of the symmetrical realization is the reduction in the number of multiplications. Since multiplication typically takes much more computer time than addition, the time savings can be significant.

Cascade Realization

The numerator of an FIR transfer function can be factored into a sequence of quadratic terms allowing for a cascade of second order sections. In this case the realization structure is shown in Figure 8.9.

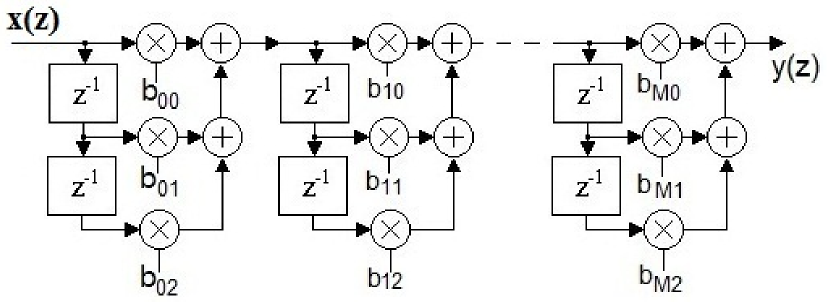


Figure 8.9
Cascaded second order sections for an FIR filter having M+1 sections.

Example 8.2
Use the MATLAB® fir1 function to create a 9th order FIR filter using a Hamming window. Set the sample frequency to 22050 Hz and the cutoff frequency to 3000 Hz. Find the coefficients for the cascade realization of this filter.

Solution
The following statements in MATLAB® create the filter and store the numerator coefficients in a vector called num.

```
fs = 22050;
fc = 3000;
N = 9;
num = fir1(N, fc/(fs/2), hamming(N+1));
```

To get the second order section coefficients we could find the roots of `num`. MATLAB® has a function called `tf2sos` that will do this in a single statement.

```
sos = tf2sos(num, 1);
[r c] = size(sos);
for i = 1:r
    for j = 1:c
        fprintf(1, '%2.6f ', sos(i, j));
    end
    fprintf(1, '\n');
end
```

This code prints the following lines:

```
-0.003974 0.021105 -0.000000 1.000000 0.000000 0.000000
 1.000000 0.811702 -0.188298 1.000000 0.000000 0.000000
 1.000000 1.677041  1.986063 1.000000 0.000000 0.000000
 1.000000 1.284306  1.000000 1.000000 0.000000 0.000000
 1.000000 0.844405  0.503509 1.000000 0.000000 0.000000
```

This is the *sos* matrix. It represents the coefficients in the second order polynomials. The left half of the matrix lists the three numerator coefficients and the right half lists the three denominator coefficients. The sections with coefficients labeled to match Figure 8.9 are given below.

Section 1 = $\frac{-.003974z+0.021105}{z}$	b_{00} = -0.003974, b_{01} = 0.021105, b_{02} = 0
Section 2 = $\frac{z^2+0.811702z-0.188298}{z^2}$	b_{10} = 1, b_{11} = 0.811702, b_{12} = -0.188298
…	…
Section 5 = $\frac{z^2+0.844405z+0.503509}{z^2}$	b_{40} = 1, b_{41} = 0.844405, b_{42} = 0.503509

The advantage of breaking an FIR filter into second order sections is that the computation can be modularized. In the example above, it would be possible to implement the filter using six small microcontrollers each doing only the computation for a single section. The modules may also be reorganized and appear in any order without changing the overall transfer function. This has some advantage in implementation when a low gain module can be placed ahead of a high gain module to help with scaling.

Parallel Realization

Consider an FIR transfer function of order six given by

$$H(z) = b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} + b_4 z^{-4} + b_5 z^{-5} + b_6 z^{-6}$$

We can rewrite this equation as two equations consisting of the even and odd terms.

$H_1(z) = b_0 + b_2 z^{-2} + b_4 z^{-4} + b_6 z^{-6}$ and

$H_2(z) = b_1 z^{-1} + b_3 z^{-3} + b_5 z^{-5} \;=\; z^{-1}(b_1 + b_3 z^{-2} + b_5 z^{-4})$

The original transfer function $H(z)$ is the sum of $H_1(z)$ and $H_2(z)$ and can be realized as shown in Figure 8.10. This parallel realization is commonly referred to as *polyphase* realization.

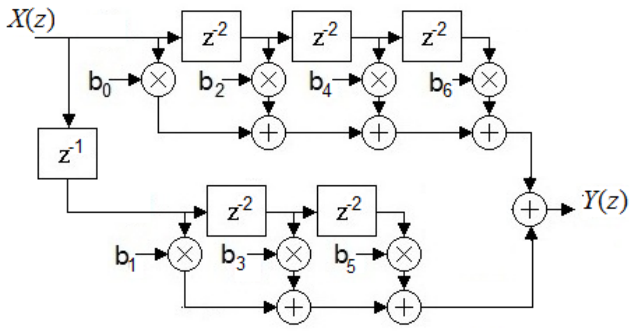


Figure 8.10
Parallel, or *polyphase* implementation of a 6th order FIR filter.

In Figure 8.10 we have just two phases but it is clear that we could use the same technique to more phases for a higher order filter. Each phase can be implemented in parallel with the others and it becomes possible to do the implementation with multiple microcontrollers, each of which is doing only a few multiplications per cycle.

Lattice Realization
Skip

IIR Structures
IIR filters can be unstable and they tend to be more sensitive to coefficient quantization and round off noise because the feedback allows any error in the system to reverberate. IIR realizations are of interest then, not only for computational efficiency but also because some structures provide better response to errors than do others. In this section we consider a direct, transposed, parallel, cascade, lattice, and state variable realizations for IIR filters.

Direct Realization
The general equation for an IIR transfer function can be written as
$$y(z) + a_1 z^{-1} y(z) + \ldots + a_N z^{-N} y(z) = b_0 x(z) + b_1 z^{-1} x(z) + \ldots + b_M z^{-M} x(z)$$
The two sides of this equation each resemble the form for an FIR filter and can be implemented separately and summed to form the output. This is shown in Figure 8.13.

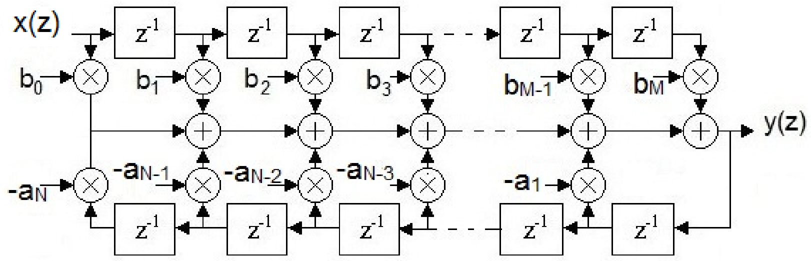


Figure 8.13
Realization of an IIR filter. This is referred to a direct form 1. This form is not canonic. In general N ≥ M.

In Figure 8.13, the numerator is of order *M* and the denominator (the feedback terms) is of order N. To be causal we must have $N \geq M$. There are *M* unit delays in the numerator and *N* unit delays in the denominator to make a total of *N*+*M*. A system is said to be *canonic* when the number of unit delays required for the realization is equal to the system order so this realization is not canonic. In order to make the system canonic, and thereby reduce the computational complexity, we need to find a way to share the unit delays.

Consider the general form of the IIR difference equation
$$y[n] = \sum_{i=0}^{M} b_i x[n-i] - \sum_{i=1}^{N} a_i y[n-i]$$
Using the shifting property of the *z*-transform we can write
$$Y(z) = X(z) \sum_{i=0}^{M} b_i z^{-i} - Y(z) \sum_{i=1}^{N} a_i z^{-i}$$
Dividing both sides by the sum of the b_i terms gives

$$\frac{Y(z)}{\sum_{i=0}^{M} b_i z^{-i}} = X(z) - \frac{Y(z)\sum_{i=1}^{N} a_i z^{-1}}{\sum_{i=0}^{M} b_i z^{-i}}$$

Create an intermediate temporary variable $T(z)$ defined as

$$T(z) = \frac{Y(z)}{\sum_{i=0}^{M} b_i z^{-i}}$$

This allows us to write the following two equations

$$Y(z) = T(z)\sum_{i=0}^{M} b_i z^{-i} \qquad (8.4)$$

$$T(z) = X(z) - T(z)\sum_{i=1}^{N} a_i z^{-i} \qquad (8.5)$$

Figure 8.14 (Left) shows the implementation of these two equations. It is clear that the left side of Figure 8.14 has duplicate delay functions and can be collapsed into the right side which represents the canonic direct realization of an IIR filter.

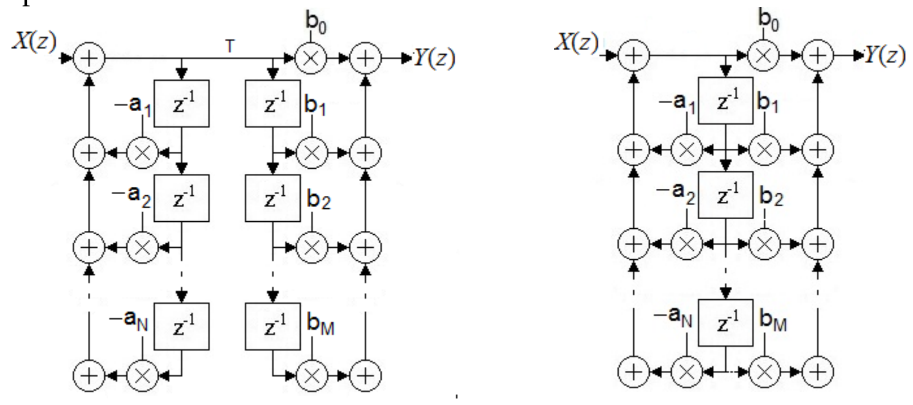


Figure 8.14
(Left) Realization of equations (8.4) and (8.5) with the intermediate variable T. (Right) Canonic direct realization for an IIR filter. In general N ≥ M.

Transposed Realization
Figure 8.15 shows the transpose realization obtained by transposing the canonic direct realization. This realization has no computational advantage over the direct form. Like its FIR counterpart, the unit delays shift product terms rather than input variables.

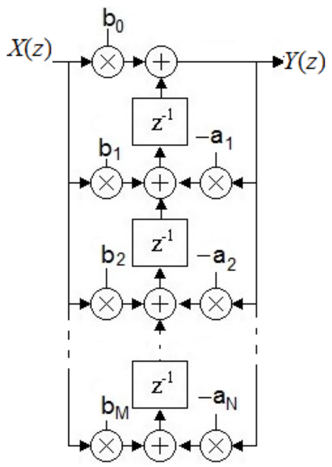


Figure 8.15
Transposed direct form for an IIR filter.

Parallel Realization
For an IIR filter the parallel realization is obtained by expressing the transfer function as a sum of terms using a partial fraction expansion. If the IIR filter has any poles at the origin, these must be removed and treated separately as delays. We can also assume that there are no repeated roots in the denominator of the transfer function which is usually the case. For a transfer function in z with a numerator of order M and a denominator of order N and $N \geq M$ and no repeated poles the partial fraction expansion can be written as

$H(z) = \dfrac{N(z)}{D(z)} = K + \sum_{i=1}^{N} \dfrac{A_i}{z - p_i}$ where p_i is a root of the denominator and A_i and K are constants.

The value of K can will be nonzero only if $M = N$ and can be obtained by direct division. The values of the A_i can be found as

$$A_i = (z - p_i)H(z)\big|_{z=-p_i}$$

If p_i is complex, A_i will also be complex and the complex term must be combined with its conjugate to form a real second order section. If $A_i = R_A + jI_A$ and $p_i = \alpha + j\beta$ the complex second order section will be given by

$$\frac{A_i}{z - p_i} + \frac{A_i^*}{z - p_i^*} = \frac{2(R_A z + \alpha R_A + I_A \beta)}{z^2 - 2\alpha z + \alpha^2 + \beta^2} \qquad (8.6)$$

Example 8.4
A fourth-order low-pass Chebyshev filter is given by

$$H(z) = 2.3972x10^{-4} \frac{z^4 + 4z^3 + 6z^2 + 4z + 1}{z^4 - 3.4256z^3 + 4.49158z^2 - 2.6639z + 0.60205}$$

Find the parallel realization equations using a partial fraction expansion.

Solution
MATLAB® has a function called `residue` which will produce the terms needed for a partial fraction expansion.

```
   num = 0.00023972*[1 4 6 4 1];
   den = [1 -3.4256 4.49158 -2.6639 0.60205];
   [A P K] = residue(num, den);
```

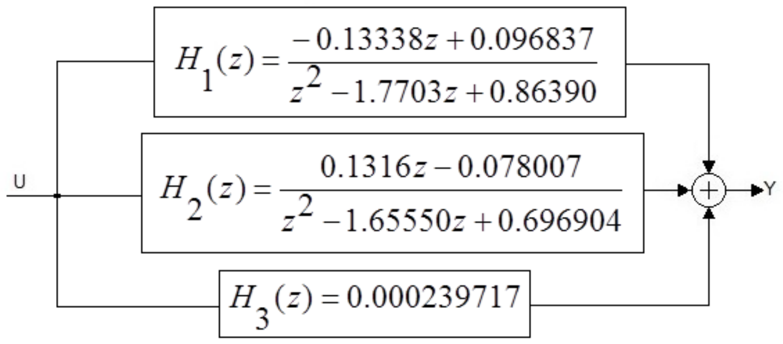
This produces a value for K as
K = 2.39717e-004

The A_i coefficients and the values of the poles are given in Table 8.3.

i	A_i	p_i
1	-0.066690 + j0.037434	0.885166 + j0.283515
2	-0.066690 - j0.037434	0.885166 - j0.283515
3	0.067580 - j0.156334	0.827749 + j0.108329
4	0.067580 + j0.156334	0.827749 + j0.108329

Table 8.3
A_i and p_i values of the partial fraction expansion of a fourth-order Chebyshev filter.

Using equation (8.6), the corresponding second order sections can be calculated. The parallel structure for this filter is shown in Figure 8.16.



$H_1(z) = \frac{-0.13338z + 0.096837}{z^2 - 1.7703z + 0.86390}$
$H_2(z) = \frac{0.1316z - 0.078007}{z^2 - 1.65550z + 0.696904}$
$H_3(z) = 0.000239717$
U
Y

Figure 8.16
Parallel realization for a fourth-order low pass Chebyshev filter.

The parallel realization has the advantage that each section can be computed independently of the others so it is possible that each section could be done by its own microcontroller. Quantization noise is also reduced and we will take this up in the next section. For the parallel structure however, the numerators and denominators fit together and cannot be interchanged. That is, if you swap the numerators of sections 1 and 2 but not the denominators you get a different filter. This is not true, as we shall see, for cascaded sections.

We note also that the partial fraction expansion can be written in a slightly different form given by

$$H(z) = \frac{N(z)}{D(z)} = K + \sum_{i=1}^{N} \frac{A_i z}{z - p_i}$$

This form is implemented in the MATLAB® `residuez` function.

Cascade Realization

The cascade or series realization for IIR filters is created by factoring the numerator and denominator polynomials of a transfer function into second order sections. The system input, x, is the input to the first section. Thereafter, each section gets its input from the previous section's output. This is illustrated in Figure 8.17.

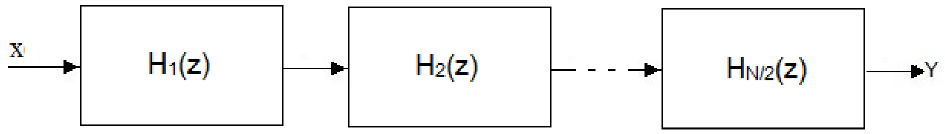


Figure 8.17
Cascaded second order sections.

To write the transfer functions for each of the second order sections, we factor the numerator and denominator into quadratic terms. If the order is odd, one section is typically added as a first order section. An advantage of cascading sections over arranging them in parallel, is that a quadratic term from the numerator can be tied to any of the quadratic terms from the denominator. This often allows arrangement of poles and zeros that are better scaled and less prone to overflow.

Example 8.5
Show how to arrange the following sixth order elliptic low pass filter in a cascaded realization of second order sections.

$$H(z)=\frac{0.04218z^6+0.03671z^5+0.09747z^4+0.0746z^3+0.09747z^2+0.03571z+0.04218}{z^6-2.32027z^5+3.5786z^4-3.25141z^3+2.04355z^2-0.75926z+0.14353}$$

Solution
We could find the roots of the numerator and denominator and match up the conjugate pairs to form second order sections. MATLAB® has a function called `tf2sos` that does this. Here is the MATLAB® code.

```
   num = [0.04218  0.03671  0.09747  0.07496  0.09747  0.03671  0.04218];
   den = [1.00000  -2.32027  3.57586  -3.25141  2.04355  -0.75926  0.14353];
   sos = tf2sos(num, den);
```

The matrix `sos` will have three rows and six columns. Each row represents a second order section. The first three columns are the numerator coefficients and the second three columns hold denominator coefficients. The contents of `sos` are shown in Table 8.4.

Section	b_0	b_1	b_2	a_0	a_1	a_2
1	0.042184	0.058486	0.042184	1.000000	-0.828188	0.253422
2	1.000000	-0.057559	1.000000	1.000000	-0.758492	0.627054
3	1.000000	-0.458728	1.000000	1.000000	-0.733589	0.903236

Table 8.4
The contents of `sos` after converting the transfer function to second order sections.

By default MATLAB® matches the pole pair closest to the unit circle with the nearest zero pair. It repeats this procedure until all pairs are matched. The scaling is placed in the first section.

Both scaling and section ordering can be changed with function options. Figure 8.18 shows the completed schematic of the second order cascaded sections.

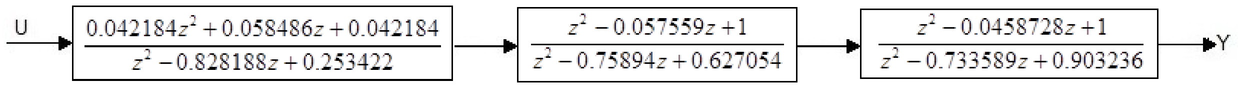

U
$\frac{0.042184z^2+0.058486z+0.042184}{z^2-0.828188z+0.253422}$
$\frac{z^2-0.057559z+1}{z^2-0.75894z+0.627054}$
$\frac{z^2-0.0458728z+1}{z^2-0.733589z+0.903236}$
Y

Figure 8.18
Cascaded second order sections for a sixth order elliptic filter.

Lattice Realization
Skip

State Space Representation
The many realization structures suggest ways in which the difference equation can be rearranged for the purpose of implementation. Normally we expect to gain some computational advantage from a given realization such as in speed, space, or error reduction. Any of the realizations presented can also be represented as a *state space system.* A state space view sees a system as a collection of unique states which can be used to completely characterize the system. We define the *state* of a system as the information needed which, along with the input, can be used to determine the output at a particular time and at any point after that time. This state is typically represented by a set of variables called the *state variables*. The state variables are not outputs, rather, the output is typically a function of one or more state variables.

The information contained in a system at some point in time is stored in the system's memory elements. For DSP systems, these memory elements are the unit delays. Hence we use the output of the unit delays as the system state variables.

Consider a second order system characterized by the difference equation
$y[n]=b_0x[n]+b_1x[n-1]+b_2x[n-2]-a_1y[n-1]-a_2y[n-2]$
This difference equation can be realized by direct form 2 as shown in Figure 8.23.

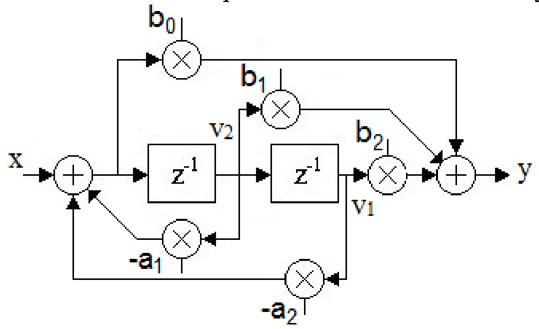


Figure 8.23
Second order direct form 2 with state variables v_1 and v_2.

We can write the following equations to characterize the system in Figure 8.23 in state variables.
$zv_1 = v_2$ (8.13)
$zv_2 = -a_2v_1 - a_1v_2 + x$ (8.14)
$y = (b_2 - b_0a_2)v_1 + (b_1 - b_0a_1)v_2 + b_0x$ (8.15)

Equations (8.13) and (8.14) are called the system state equations and equation (8.15) is called the system output equation. In general, for an Nth order system with one input and one output there will be N state equations and one output equation. Note that this representation presents the same information as does the difference equation since we can always solve one equation for v_1 and substitute its value into the other two. We can then solve one of the other two equations for v_2 and substitute it into the remaining equation to get the original difference equation.

In matrix form the state equations and the output equation can be written as

$z\mathbf{v} = \mathbf{A}\mathbf{v} + \mathbf{B}x$ (8.16)

$y = \mathbf{C}\mathbf{v} + \mathbf{D}x$ (8.17)

Where

$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$ and

$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -a_2 & -a_1 \end{bmatrix}$, $\mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $\mathbf{C} = \begin{bmatrix} (b_2 - b_0 a_2) & (b_1 - b_0 a_1) \end{bmatrix}$, and $\mathbf{D} = [1]$ (8.18)

For an *N*th order system, instead of an *N*th order difference equation, we have *N* state equations and an output equation. The **A** matrix will be *N* x *N*, **B** will be *N* x 1, **C** will be 1 x *N*, and **D** will be 1 x 1.

To arrive at the transfer function in matrix form solve equation (8.16) for **v.**

$\mathbf{v} = (z\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}x$

Substitute this value for **V** into equation (8.17) to get

$y = [\mathbf{C}(z\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}]x$

Define the transfer function as

$H(z) = [\mathbf{C}(z\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}]$ (8.19)

Example 8.7

Use equation (8.19) to find the transfer function for a digital filter that has the following state space representation.

$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -0.9 & 1.2 \end{bmatrix}$, $\mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $\mathbf{C} = \begin{bmatrix} (0.1) & (3.2) \end{bmatrix}$, and $\mathbf{D} = [1]$ (8.18)

Solution

$$H(z) = \begin{bmatrix} 0.1 & 3.2 \end{bmatrix} \begin{bmatrix} z & -1 \\ 0.9 & z-1.2 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 1 \end{bmatrix} + [1]$$

$$H(z) = \begin{bmatrix} 0.1 & 3.2 \end{bmatrix} \frac{\begin{bmatrix} z\text{-}1.2 & 1 \\ -0.9 & z \end{bmatrix}}{|z\mathbf{I} - \mathbf{A}|} \begin{bmatrix} 0 \\ 1 \end{bmatrix} + 1$$

$$H(z) = \frac{z^2 + 2z + 1}{z^2 - 1.2z + 0.9}$$

We note that the denominator is given by $|z\mathbf{I} - \mathbf{A}| = z^2 - 1.2z + 0.9$ and the equation

$|z\mathbf{I}-\mathbf{A}|=0$ is called the *characteristic equation*, the roots of which determine the system stability.

If we take the transpose of the matrix form of the transfer function we get the transfer function for the transposed system. Let the transpose of a matrix be designated with a T superscript as in $\mathbf{A}^{\mathbf{T}}$. Noting that the transpose of three multiplied matrices is given by:

$$(\mathbf{XYZ})^{\mathrm{T}}=(\mathbf{Z}^{\mathrm{T}}\mathbf{Y}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}})$$

We get

$$\text{Transpose}\,(\mathrm{H(z)})=[\mathbf{B}^{\mathrm{T}}(z\mathbf{I}-\mathbf{A}^{\mathrm{T}})^{-1}\mathbf{C}^{\mathrm{T}}+\mathbf{D}^{\mathrm{T}}]$$

Comparing this equation to equation (8.19) we see that we can represent the transposed system by transposing each matrix and swapping the transposed **B** matrix with the transposed **C** matrix. The transposed system matrix equations are given by

$$\mathbf{v}=\begin{bmatrix}v_1\\v_2\end{bmatrix} \text{ and}$$

$$\mathbf{A}=\begin{bmatrix}0 & -a_2\\1 & -a_1\end{bmatrix},\ \mathbf{B}=\begin{bmatrix}b_2-b_0a_2\\b_1-b_0a_1\end{bmatrix},\ \mathbf{C}=\begin{bmatrix}0 & 1\end{bmatrix}, \text{ and } \mathbf{D}=[1] \qquad (8.20)$$

It is easy to verify that this is the set of state variable equations that we would get if we assigned state variables to the transposed system of Figure 8.15. This mathematical derivation of the transposed system high lights one of the advantages of the state space representation, namely, it is possible to create many numerically different representations for the same system and some of these will have computational advantages and result in less numerical error.

To rewrite the state equations with other values consider equations (8.16) and (8.17) when the variable x undergoes a coordinate transformation. Toward this end we define a new state variable q such that

$$\mathbf{Q}=\mathbf{T}^{-1}\mathbf{v}$$

Where the matrix **T** is an *N* x *N* transformation matrix.

Equations (8.16) and (8.17) become

$$z\mathbf{TQ}=\mathbf{ATQ}+\mathbf{B}x \qquad (8.21)$$

$$y=\mathbf{CTQ}+\mathbf{D}x \qquad (8.22)$$

Multiplying equation (8.21) by $\mathbf{T}^{-1}$ gives

$$z\mathbf{Q}=\mathbf{T}^{-1}\mathbf{ATQ}+\mathbf{T}^{-1}\mathbf{B}x \qquad (8.23)$$

$$y=\mathbf{CTQ}+\mathbf{D}x \qquad (8.24)$$

Comparing (8.23) and (8.24) with (8.16) and (8.17), we see that we can do a coordinate transformation if we define a somewhat arbitrary non-singular *N* x *N* matrix **T** and we make the following matrix replacements.

$$\mathbf{A}\leftarrow\mathbf{T}^{-1}\mathbf{AT} \qquad (8.25)$$

$$\mathbf{B}\leftarrow\mathbf{T}^{-1}\mathbf{B} \qquad (8.26)$$

$$\mathbf{C}\leftarrow\mathbf{CT} \qquad (8.27)$$

$$\mathbf{D}\leftarrow\mathbf{D} \qquad (8.28)$$

Example 8.8

For the system described by the state matrices in Example 8.7, find the transformed state matrices using the transformation matrix given by

$$\mathbf{T} = \begin{bmatrix} -1.3608 & 0 \\ -.8165 & 1 \end{bmatrix}$$

Show that, in general, the transfer function is unchanged by coordinate transformations.

Solution

The original **A, B, C**, and **D** matrices are:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -0.9 & 1.2 \end{bmatrix},\ \mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix},\ \mathbf{C} = \begin{bmatrix} (0.1) & (3.2) \end{bmatrix}, \text{ and } \mathbf{D} = [1]$$

Using equations (8.25) through (8.28) we have

$$\mathbf{A} \leftarrow \mathbf{T}^{-1}\mathbf{A}\mathbf{T} = \begin{bmatrix} -1.3608 & 0 \\ -.8165 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 & 1 \\ -0.9 & 1.2 \end{bmatrix} \begin{bmatrix} -1.3608 & 0 \\ -.8165 & 1 \end{bmatrix} = \begin{bmatrix} 0.6 & -0.7348 \\ 0.7348 & 0.6 \end{bmatrix}$$

$$\mathbf{B} \leftarrow \mathbf{T}^{-1}\mathbf{B} = \begin{bmatrix} -1.3608 & 0 \\ -.8165 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\mathbf{C} \leftarrow \mathbf{C}\mathbf{T} = \begin{bmatrix} (0.1) & (3.2) \end{bmatrix} \begin{bmatrix} -1.3608 & 0 \\ -.8165 & 1 \end{bmatrix} = \begin{bmatrix} -2.7489 & 3.2 \end{bmatrix}$$

$$\mathbf{D} \leftarrow \mathbf{D} = [1]$$

To show that in general, the transfer function remain unchanged we begin with equation (8.19).

$$H(z) = [\mathbf{C}(z\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}]$$

Substituting the transformed versions of the matrices into this equation gives

$$H_T(z) = [\mathbf{CT}(\mathbf{zI} - \mathbf{T}^{-1}\mathbf{AT})^{-1}\mathbf{T}^{-1}\mathbf{B} + \mathbf{D}]$$

This equation can be written as

$$H_T(z) = [\mathbf{CTT}^{-1}(\mathbf{zI} - \mathbf{A})^{-1}\mathbf{TT}^{-1}\mathbf{B} + \mathbf{D}]$$

Since $\mathbf{TT}^{-1} = \mathbf{I}$ we have

$$H_T(z) = [\mathbf{C}(z\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}] = H(z)$$