Chapter

4

Digital Filters

In this chapter we study the implementation of digital filters specifically useful for mixed-signal circuit design. As in the last chapter, our filters will be based on the integrator and, to a lesser extent, the differentiator. Prior to studying the material in this chapter the reader may want to review Sec. 1.2.

4.1 SPICE Models for DACs and ADCs

In order to verify, using SPICE simulations, the theory for digital filters discussed in this chapter, we start out by developing simulation models for ideal digital-to-analog converters (DACs) and analog-to-digital converters (ADCs). Our goal is to generate ideal DACs and ADCs that we can place in a mixed-signal simulation to either generate a digital word based on an analog input or look at the spectrum of a digital signal.

4.1.1 The Ideal DAC

Consider the ideal transfer characteristics of a 3-bit DAC shown in Fig. 4.1. Notice in this figure that we have drawn two reference voltages, V_{REF+} and V_{REF-}, and are assuming that $V_{REF+} > V_{REF-}$. When a digital input of 000 is applied to the DAC, the output voltage becomes V_{REF-}. When the input code is increased to 001, the output of the DAC (an analog voltage defined at discrete amplitude levels) increases by one least significant bit (LSB). If the DAC has an input code with a number of bits, N, then we can define an LSB as

$$1 \text{ LSB} = \frac{V_{REF+} - V_{REF-}}{2^N} = V_{LSB} \quad \text{for} \quad N \geq 2 \tag{4.1}$$

If, for example, $V_{REF+} = 1.25$ V and $V_{REF-} = 0.25$ V and $N = 3$, then our LSB, the vertical distance between adjacent points in Fig. 4.1, is 0.125 V. Note that in our discussion of an ideal DAC we are assuming that the output of the DAC ranges from V_{REF-} up to $V_{REF+} - 1$ LSB. We could just as easily have assumed that the output ranged from $V_{REF-} + 1$ LSB up to V_{REF+}. The important thing to notice is that the DAC output range is 1 LSB smaller than the difference between the positive and negative reference voltages. For the DAC developed in this chapter, we will assume $V_{REF+} = VDD = 1.0$ V and $V_{REF-} = 0$ V. Selection of the power supply rails, which are noise free in a SPICE

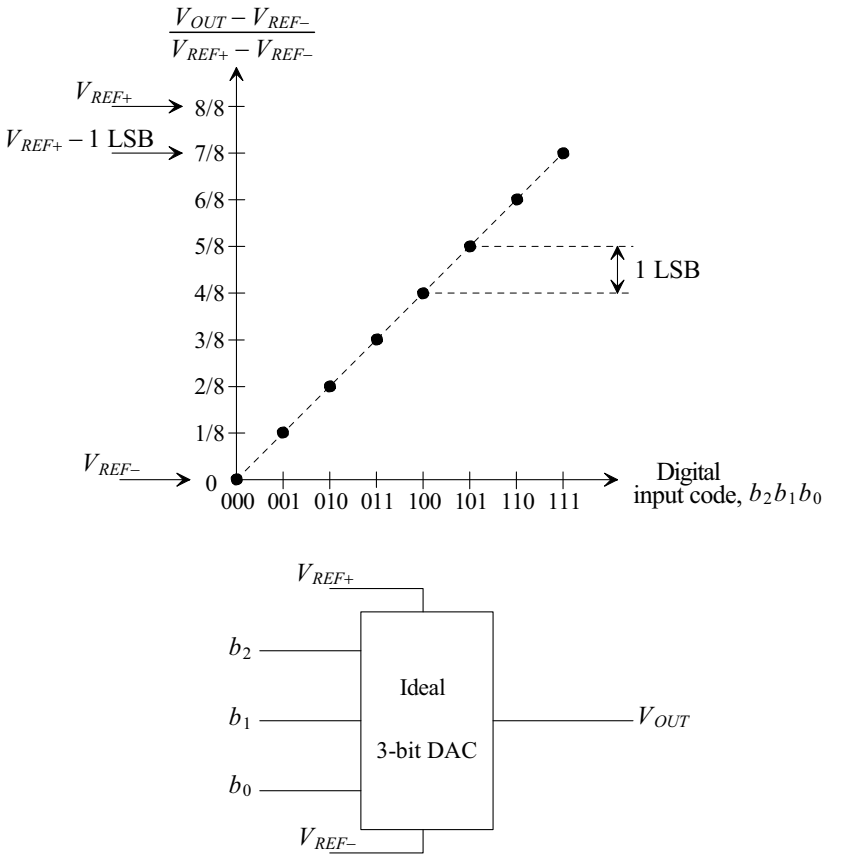


Figure 4.1 An ideal 3-bit DAC.

simulation, allow the maximum output range for the DAC (assuming the reference voltages are indeed the maximum and minimum voltages in the system, i.e., no charge pumps or external, larger, power supply voltages). If we need more resolution when using our ideal DAC, we will simply increase the number of bits, *N*, used and thus decrease the value of the DAC's LSB.

SPICE Modeling the Ideal DAC

We can write the output of the ideal DAC in terms of the reference voltages and digital input codes b_N (which are logic "0" or "1"), and assuming that an input code of all zeroes results in an output voltage of V_{REF-}, as

$$V_{OUT} = (V_{REF+} - V_{REF-}) \cdot \left(\frac{b_{N-1}}{2^1} + \frac{b_{N-2}}{2^2} + ... + \frac{b_1}{2^{N-1}} + \frac{b_0}{2^N} \right) + V_{REF-} \quad (4.2)$$

or

$$V_{OUT} = (V_{REF+} - V_{REF-}) \cdot \frac{1}{2^N} \cdot (b_{N-1}2^{N-1} + b_{N-2}2^{N-2} + ... + b_1 \cdot 2^1 + b_0) + V_{REF-} \qquad (4.3)$$

We can implement this equation, in SPICE, using a nonlinear dependent source (a B source). For a 3-bit, ideal DAC, the statement that implements this equation may look like

```
*Nonlinear dependent source, B, for generating the 3-bit DAC output
Bout Vout 0 V=((v(vrefp)-v(vrefm))/8)*(v(B2L)*4+v(B1L)*2+v(B0L))+v(vrefm)
```

The terms BXL correspond to logic signals that have a value of 1 V or 0 V.

Example 4.1
Write the nonlinear dependent SPICE source statement for an ideal 12-bit DAC.

The statement follows:

```
Bout Vout 0 V=((v(vrefp)-v(vrefm))/4096)*
+(v(B11L)*2048)+v(B10L)*1024+v(B9L)*512+v(B8L)*256+
+v(B7L)*128+v(B6L)*64+v(B5L)*32+v(B4L)*16+v(B3L)*8+
+v(B2L)*4+v(B1L)*2+v(B0L))+v(vrefm)
```

remembering that a "+" in the first column of a line indicates that the text on the remainder of the line behaves as if it were typed at the end of the previous line. It doesn't indicate addition. ■

The next thing we need to concern ourselves with is the digital logic levels. We want to use our ideal DAC with real circuits where the logic voltage levels may not be well defined. We need to determine and use a switching-point voltage based on the power-supply voltage *VDD*. We will assume the input logic code is a valid logic "1" if its amplitude is greater than *VDD*/2 and a logic "0" if its amplitude is less than *VDD*/2. The switch implementation used to generate the logic signals (1 V and 0 V) used in our dependent source from real signals is shown in Fig. 4.2.

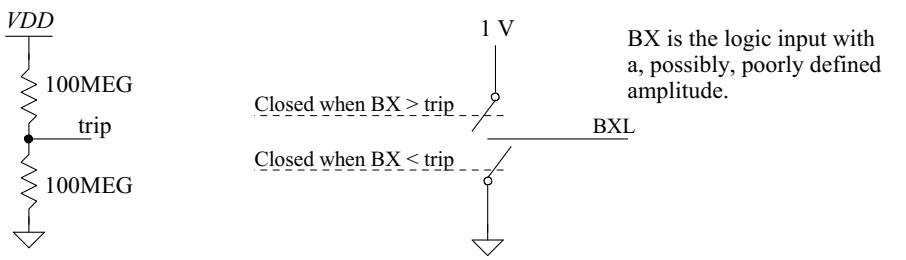


Figure 4.2 Generating logic levels using voltage-controlled switches.

4.1.2 The Ideal ADC

The characteristics of our ideal ADC are shown in Fig. 4.3. Notice that the transfer curve is shifted to the left. If we were to flip the curve on its side and mark, with black dots, the intersection of the analog input voltage with the ADC transfer curve, we would have the DAC transfer curve of Fig. 4.1. Again, 1 LSB is given by Eq. (4.1). Notice how

converting a (normalized) input voltage of 0.1 V will result in an output code of 000 which is the same output code resulting from converting 0 V. Unlike the ideal DAC, the ideal ADC quantizes its input with the practical result of adding noise to the input signal. This noise is called *quantization noise*.

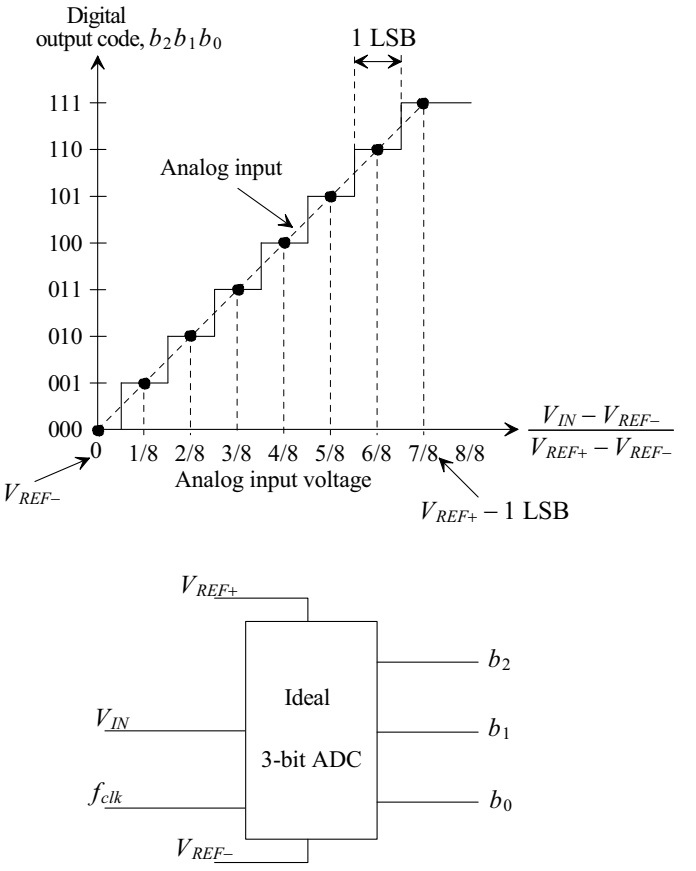


Figure 4.3 An ideal 3-bit ADC.

The implementation of the ideal ADC consists of an ideal S/H followed by passing the output of the S/H (the held signal) through an algorithm to generate the output bits. The algorithm we use is based on a pipeline ADC and follows:

1. The input signal is sampled and held.
2. This held signal is input to a comparator that compares the input value to a reference voltage.
3. If the input signal is greater than the reference voltage, the output bit is set to a high, and the reference signal is subtracted from the input. The difference is multiplied by two and passed to the output of the stage.

4. If the input signal is less than the reference voltage, the output bit is set low. The input signal is multiplied by two and passed to the output of the stage.

5. This output is used as the input to the next stage and steps 2, 3, and 4 above are repeated. This continues for N stages (where N is the number of bits in the ADC).

The reference voltage, or common mode voltage V_{CM}, can be determined by calculating the midpoint between V_{REF+} and V_{REF-} followed by subtracting V_{REF-} so that the V_{CM} is referenced to 0 V. This can be written as

$$V_{CM} = \frac{V_{REF+} + V_{REF-}}{2} \rightarrow V_{CM0} = \frac{V_{REF+} + V_{REF-}}{2} - V_{REF-} = \frac{V_{REF+} - V_{REF-}}{2} \quad (4.4)$$

We also want to level-shift the input signal so that it is referenced to 0 V. In addition, we want to shift the transfer curves to the left by 1/2 LSB as seen in Fig. 4.3. In order to do this we use the following SPICE statement (for an 8-bit ADC where V(OUTSH) is the output voltage of the ideal S/H [the input to the pipeline algorithm above])

```
* Level shift by VREFM and 1/2LSB
BPIP PIPIN 0 V=V(OUTSH)-V(VREFM)+((V(VREFP)-V(VREFM))/2^9)
```

The last term in this statement is 1/2 LSB, which is given by

$$1/2 \text{ LSB} = \frac{V_{REF+} - V_{REF-}}{2^{N+1}} \text{ assuming } V_{REF+} > V_{REF-} \geq 0 \quad (4.5)$$

We are level-shifting the input and common-mode voltage because we want to make the model as flexible as possible. For example, we want the ADC model to function if V_{REF+} = 0.5 V and V_{REF-} = 0.25 V. Note that if $V_{REF-} = 0$ and $V_{REF+} = VDD$, the model can be simplified.

4.1.3 Number Representation

Suppose we have an ideal ADC and DAC each with a resolution of 8-bits, a V_{REF+} = 1 V and a V_{REF-} of ground. Using Eq. (4.1) the data converter's LSB is 3.906 mV. The minimum output of the DAC, Fig. 4.1, is ground and the maximum output is VDD – 1 LSB or 0.9961 V. An input sinewave swinging from ground to VDD and centered around the common-mode voltage, V_{CM}, of 500 mV is seen in Fig. 4.4. Also seen are the corresponding digital codes and voltages. This number format is referred to as *binary offset* format. The output of our ideal ADC and the input of our ideal DAC are in this format. A more useful format, for adding and subtracting digital numbers, is the *two's complement* format, Fig. 4.5. We get this format by complementing the MSB of a word in

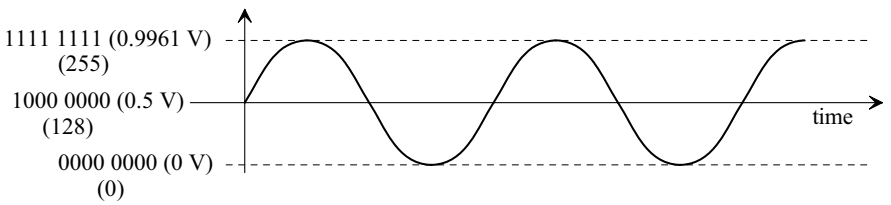


Figure 4.4 Representing a sinusoid in binary offset format.

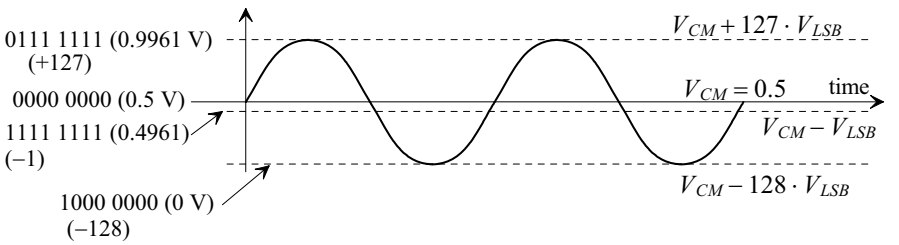


Figure 4.5 Representing a sinusoid in two's complement format.

binary offset (and thus to go back-and-forth between two's complement and binary offset we simply complement the MSB). Note, in Fig. 4.5, that −1 is represented using 1111 1111, −2 is represented using 1111 1110, etc. Also note that the MSB corresponds to the word's sign-bit. An MSB of 0 indicates a positive value (or the common-mode voltage, V_{CM}, 0000 0000) while an MSB of 1 indicates a negative value (a code or voltage below V_{CM}).

Increasing Word Size (Extending the Sign-Bit)

Often as we design digital filters we'll need to increase the word size to increase the resolution of the signal or to avoid register overflow. Figure 4.6 shows how we would increase the binary offset representation of the output of a 3-bit ADC to 8-bits while changing the format to two's complement representation. The MSB of the ADC's output is inverted and extended. Notice what would happen if our ADC's input signal were V_{CM}. The ADC's output is then 100. This is changed into 0000 0000. The common-mode signal, as seen in Fig. 4.5, can be thought of as a reference, or zero, level. Since we'll be using two's complement numbers throughout the rest of the book the reader should spend some time reviewing, and ensuring they understand, Figs. 4.4, 4.5, and 4.6.

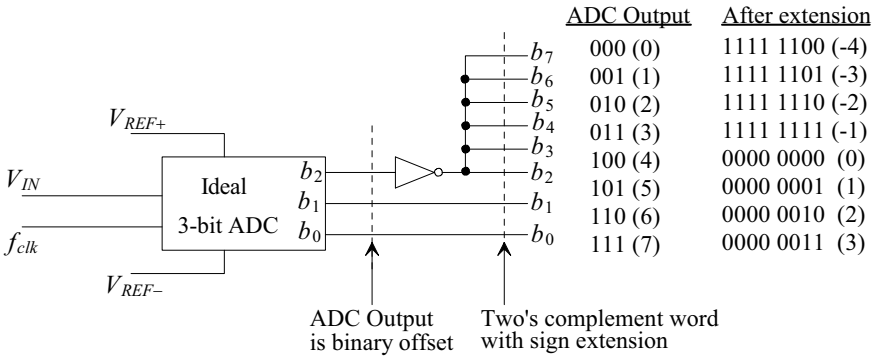


Figure 4.6 Showing how to change the output of an ADC to two's complement and how to extend the sign bit.

Example 4.2
Show how to convert a 1-bit binary offset number into 2-bit and 4-bit two's complement numbers.

A 1-bit binary offset number has values of 0 or 1 (common for the output of a noise-shaping ADC). To change this into a 2-bit two's complement number we'll represent 0 as –1 and 1 as +1 or

$$0 \rightarrow 11\ (-1)$$

$$1 \rightarrow 01\ (+1)$$

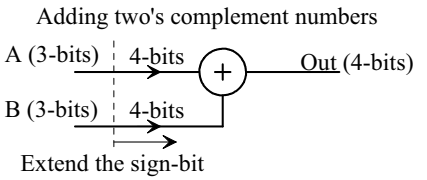
For 4-bit representations we can write $0 \rightarrow 1111\ (-1)$ and $1 \rightarrow 0001\ (+1)$. We'll use these results frequently when discussing noise-shaping (delta-sigma) data converters. ■

A comparator is an example of an ADC (or quantizer) that will generate the 1-bit number used in this example. Reviewing Eq. (4.1) we see that this equation doesn't work for determining the LSB of a 1-bit ADC. We can modify Eq. (4.1) for this situation

$$1\ \text{LSB} = V_{REF+} - V_{REF-} = V_{LSB} \text{ for } N = 1 \tag{4.6}$$

Adding Numbers and Overflow

Figure 4.7 shows how two's complement numbers are added. Note that the first thing we do, when adding two digital signals, is to extend our sign bit to avoid harmful overflow (see the allowable overflow examples in the figure). This increases the word size and allows the final output word size to always be large enough to accommodate the sum of the inputs. *Note that this is important.* In filters employing feedback, like the digital integrator seen in Fig. 1.26, we may increase the word size even more to avoid, or delay, harmful overflow. For an integrator a DC input will always, after some time, result in overflow (unless the integrator's input is always zeroes). Also note that harmful overflow is easy to detect since it will only occur when the two inputs are the same polarity (positive or negative). If the MSB of the two input words is a 0 (1) then the output word's MSB must be 0 (1) . If it's a 1 (0) then we know overflow occurred. In simpler terms, if the two inputs are positive (negative) then the output must be positive (negative).



A	B	Out (A+B)
0011 (+3)	0011 (+3)	0110 (+6)
1100 (-4)	1100 (-4)	1000 (-8) OF!
1111 (-1)	0001 (+1)	0000 (0) OF!
0011 (+3)	1111 (-1)	0110 (+2) OF!
0001 (+1)	1100 (-4)	1101 (-3)

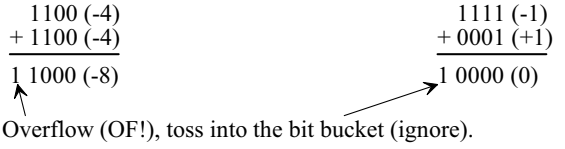
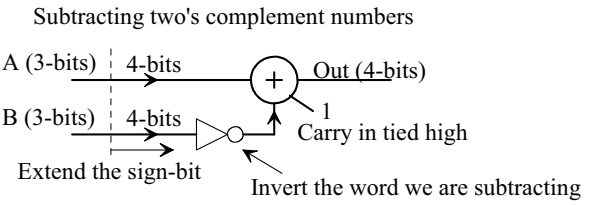


Figure 4.7 Showing how two's complement numbers are added.

Subtracting Numbers in Two's Complement Format

Figure 4.8 shows how two's complement numbers are subtracted. Here we are subtracting the input B from the input A. In order to do this we complement B (run the *N*-bit word through *N* inverters). We then tie the carry input of the adder high. Shown in Fig. 4.8 are several examples of subtraction. Note, again, how we extended the sign bit to ensure no harmful overflow occurs.



A	B	$\overline{B}+1$	Out (A-B)
0011 (+3)	1100 (-4)	0100 (+4)	0111 (+7)
1100 (-4)	1100 (-4)	0100 (+4)	0000 (0) OF!
1111 (-1)	0001 (+1)	1111 (-1)	1110 (-2) OF!
0011 (+3)	1111 (-1)	0001 (+1)	0100 (+4)
0010 (+2)	0001 (+1)	1111 (-1)	0001 (+1) OF!

Figure 4.8 Showing how two's complement numbers are subtracted.

4.2 Sinc-Shaped Digital Filters

Perhaps the most useful digital filters that we'll encounter when doing mixed-signal circuit design have Sinc-shaped responses. Let's start this section by discussing one of the simplest of these filters.

4.2.1 The Counter

Figure 4.9 shows how a counter can be used as a digital filter with a 1-bit input. If we assume the clock frequency is f_s (= $1/T_s$) and the counter is read out and reset every KT_s seconds then a constant input of 0s (–1 in two's complement, see Fig. 6.3)) will result in a counter output of 0 (–K) at the end of the time interval KT_s seconds. A constant input of logic 1s results in a counter output of *K*. An input of alternating 1s and 0s (the input is a 50% duty cycle squarewave with a frequency of $f_s/2$) results in an output of *K*/2 (0 for two's complement numbers). The output of the counter is related to its inputs using

$$y[Ki \cdot T_s] = \sum_{n=K(i-1)}^{K\cdot i-1} x[n \cdot T_s] \tag{4.7}$$

This equation simply indicates that we are taking *K* inputs, adding them together, and the result is the output of the filter (so we can think of this filter, **like any lowpass filter**, as an *averaging filter*). Rewriting Eq. (4.7) in the *z*-domain,

$$H(z) = \frac{Y(z)}{X(z)} = \sum_{n=0}^{K-1} z^{-n} = 1 + z^{-1} + z^{-2} + \ldots + z^{1-K} \tag{4.8}$$

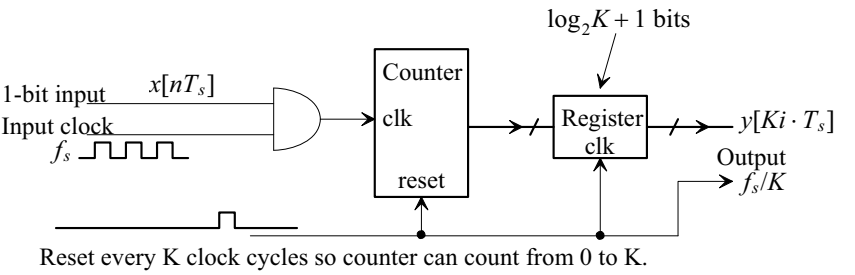


Figure 4.9 Using the counter as a digital filter.

or

$$H(z) = \frac{1 - z^{-1}}{1 - z^{-1}} \cdot (1 + z^{-1} + z^{-2} + ... + z^{1-K}) \tag{4.9}$$

The transfer function for the counter is then

$$H(z) = \frac{1 - z^{-K}}{1 - z^{-1}} \tag{4.10}$$

We'll see this equation frequently so let's spend some time on it. Note that the counter employs decimation, see Sec. 2.1.2, since the input word rate is K times faster than the output word rate. More on this in a moment (since aliasing will be a concern). The magnitude of the frequency response of Eq. (4.10) is given by

$$|H(f)| = \left| \frac{1 - e^{-j \cdot 2\pi K \frac{f}{f_s}}}{1 - e^{-j \cdot 2\pi \cdot \frac{f}{f_s}}} \right| = \frac{\sqrt{2(1 - \cos 2\pi K \cdot \frac{f}{f_s})}}{\sqrt{2(1 - \cos 2\pi \cdot \frac{f}{f_s})}} = \left| \frac{\sin\left(\pi K \cdot \frac{f}{f_s}\right)}{\sin\left(\pi \cdot \frac{f}{f_s}\right)} \right| \tag{4.11}$$

or

$$|H(f)| = K \cdot \left| \frac{Sinc\left(\pi \frac{K \cdot f}{f_s}\right)}{Sinc\left(\pi \frac{f}{f_s}\right)} \right| \tag{4.12}$$

Figure 4.10 shows the frequency response of the counter (see, also, Fig. 2.29). Note that for large K and small frequencies this equation can be approximated using

$$|H(f)| \approx K \cdot \left| Sinc\left(\pi \frac{K \cdot f}{f_s}\right) \right| \tag{4.13}$$

Figures 2.30 and 2.31 in Ch. 2 relate the amount of attenuation and droop we can expect from a Sinc response filter for various values of K. Again, these equations can be used to characterize all of the Sinc filters in this section.

Aliasing

Note, again, if we input a DC value of constant 1s then the counter's output is K (see Fig. 4.10). If our input is 101010... or 11001100... or 111000111000 etc. then the counter output is $K/2$. However, looking at the filter response seen in Fig. 4.10 and knowing an input of 101010 is a squarewave with a frequency of $f_s/2$, we would expect the counter's

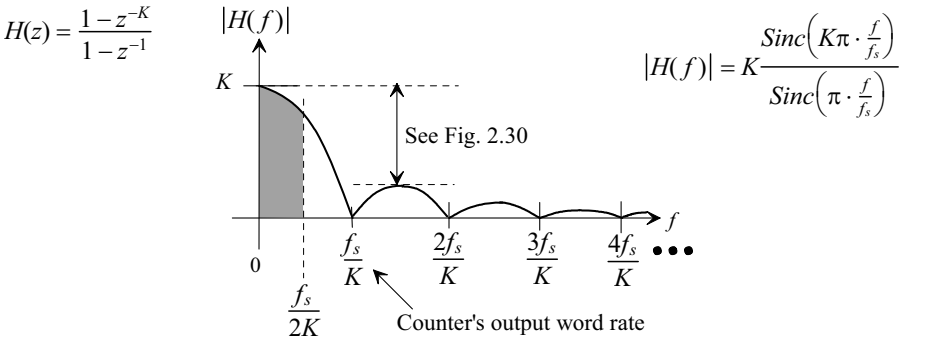


Figure 4.10 Frequency response of a Sinc-shaped digital filter.

output to be zero (and it is if we think in terms of two's complement numbers). However, let's look at the operation in terms of binary offset numbers.

Let's assume K is 8 and the counter is clocked at a frequency of 100 MHz. The 1-bit word rate coming into the counter is at 100 MHz while the counter is read out every 12.5 MHz (decimation by 8). Figure 4.11a shows the frequency response of the counter and the first-harmonic of the input of 1010101... or a squarewave at 50 MHz. We only concern ourselves with the first harmonic of the squarewave to keep the figures from getting too cluttered. A sinewave at 50 MHz, see Eqs. (1.85), having a peak-to-peak amplitude of 1, or a peak amplitude of 0.5, is represented at ± 50 MHz with dirac-delta functions having amplitudes of 0.25. Figure 4.11b shows the input signal spectrum after resampling at the output rate of 12.5 MHz (note the aliasing of the sampled signal at DC). In (c) we see that after including the counter's Sinc-shape response all of the tones disappear except for the aliased tone at DC (and tones at 100 MHz, 200 MHz, etc.) The output of the counter, which is read out as the counter is reset every KT_S seconds, (that is we don't look at the output of the counter while it's changing) is a constant value of 4 (that never changes) even though the input is a 50 MHz squarewave.

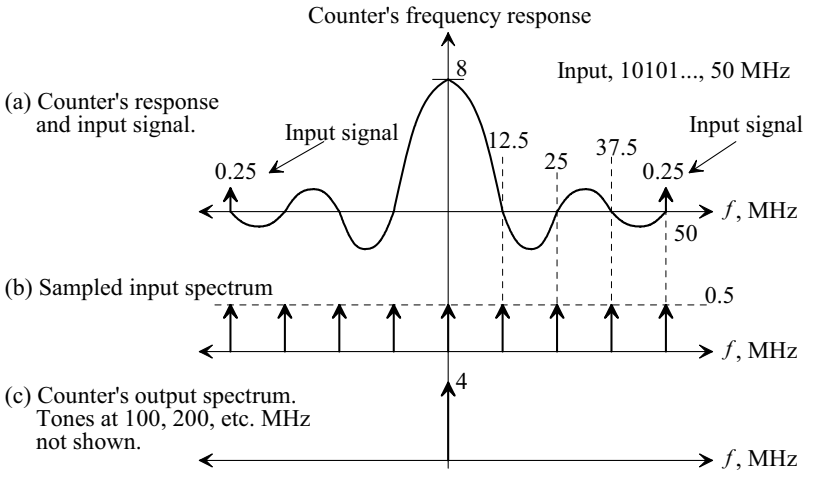


Figure 4.11 Showing how decimating using a counter results in aliasing.

Using a counter as a digital filter, as we've just seen, has limited uses because of the significant amount of aliasing that occurs with the inherent decimation. Note that we didn't discuss problems that occur when the input signal contains noise. One of the counter's uses, however, is in applications where the desired signal is constant (DC) such as some sensing applications. By using really large values of K the bandwidth can be shrunk down to limit the effects of noise while at the same time amplifying signals at DC (review Fig. 4.10).

The Accumulate-and-Dump

The counter's input was a 1-bit word. For N-bit input words the accumulate-and-dump can be used, Fig. 4.12, in place of a counter. The input signal is summed in a register for K clock cycles (note that this is the non-delaying integrator, also known as an *accumulator*, seen in Fig. 1.26). At the end of this time the sum is clocked into an output hold register and the summing register is reset (the integrator is reset). The equations and filtering behavior of the accumulate-and-dump are exactly the same as the counter. The maximum input word's value is $2^N - 1$ (all N bits high). In order to accommodate the summation of K, N-bit, input words the register size used must be at least $N + \log_2 K$ bits wide (rounding up to the nearest integer). The frequency response of the accumulate-and-dump filter or the counter at DC is K.

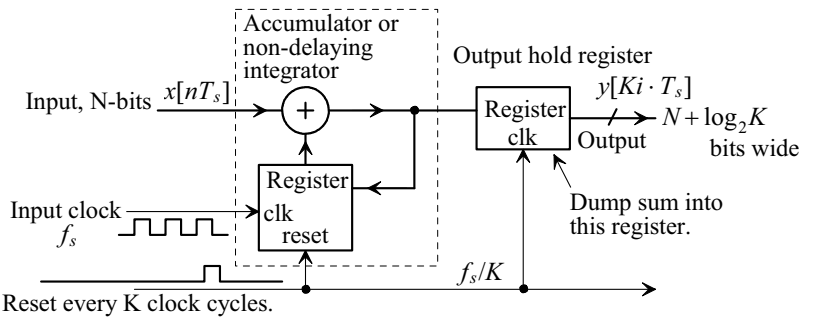


Figure 4.12 The accumulate and dump.

4.2.2 Lowpass Sinc Filters

Reviewing Eq. (4.10) one may wonder if we could implement this lowpass Sinc-response filter without the decimation inherent in a counter or an accumulate-and-dump. Figure 4.13 shows how cascading a comb filter with an integrator implements Eq. (4.10). Also seen in this figure are several z-plane plots and frequency responses for varying values of K. Note how, to get the lowpass response, we simply *cancel the zero* at DC (if this doesn't intuitively make sense review Sec. 1.2.3). We'll show that to implement bandpass or highpass Sinc-response filters all we do is cancel a zero using a pole at some other point on the unit circle. Notice in Fig. 4.13 that we've defined the bandwidth of the desired signal (where the droop is −3.9 dB, see Figs. 2.31 or 4.10) using

$$B = \frac{f_s}{2K} \tag{4.14}$$

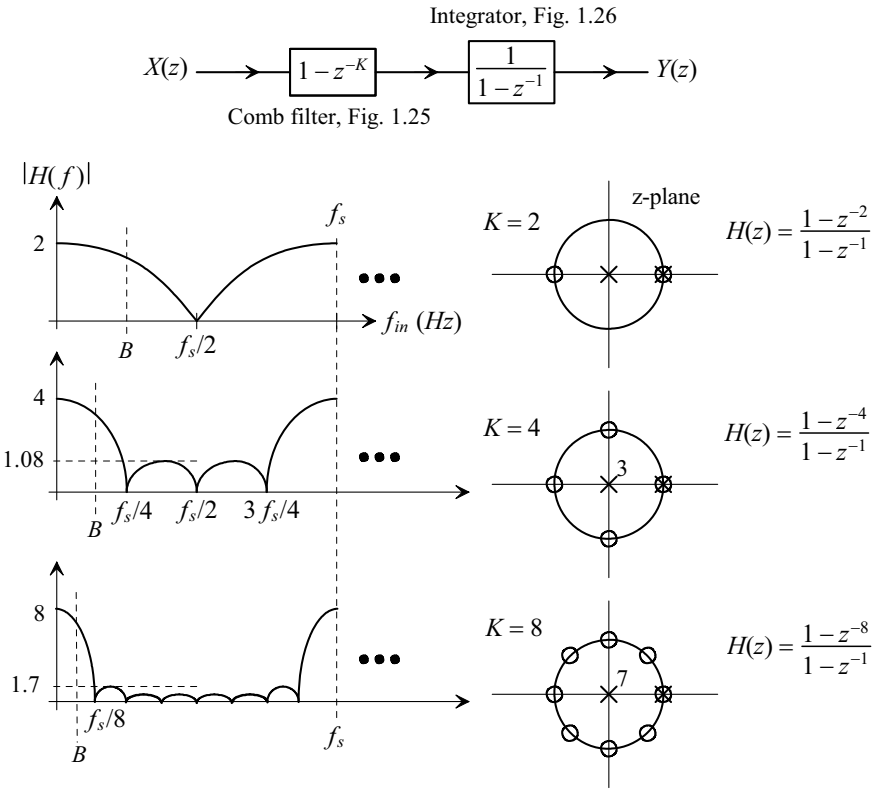


Figure 4.13 Lowpass Sinc-response filters with varying values of K.

Example 4.3

Assuming an 8-bit input word sketch the implementation of the Sinc-shaped filter

$$\frac{1-z^{-8}}{1-z^{-1}}$$

What is the output word size? Using SPICE verify that the filter's frequency response is given by Eq. (4.13) with $K = 8$. Assume $f_s = 100$ MHz.

The output of the filter is the sum of K, N-bit, words so the filter's output word size must grow to

$$\text{Output word size} = N + \log_2 K \tag{4.15}$$

For this example where $K = 8$ and the input word size is 8-bits the final output word must grow to 11-bits. The filter is sketched in Fig. 4.14.

Figure 4.15 shows the filter's input and output when the input frequency is 6.25 MHz ($= f_s/2K$). As seen in Figs. 4.10 and 2.31 the filter's attenuation is –3.9 dB or 0.638. The phase shift is – 78.75 degrees (see Eqs. [1.54] and [1.63]). ■

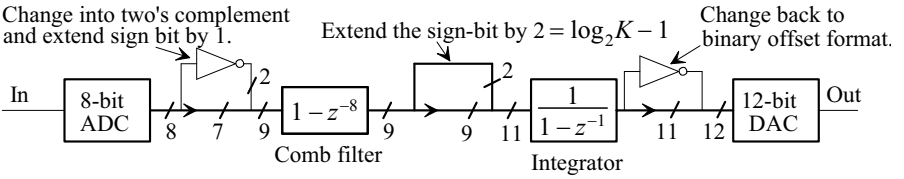


Figure 4.14 Digital filter sketch for Ex. 4.3.

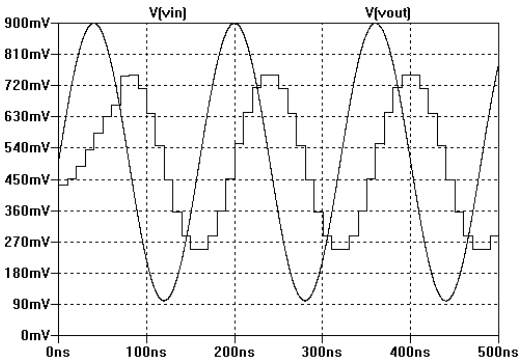


Figure 4.15 Input and output of the filter in Fig. 4.14 at a frequency of 6.25 MHz.

Example 4.4

Determine, and sketch, the time-domain impulse response of an averaging filter with $K = 8$.

The transfer function of the filter is given by Eq. (4.10) or

$$H(z) = \frac{1 - z^{-8}}{1 - z^{-1}} = 1 + z^{-1} + z^{-2} + z^{-3} + z^{-4} + z^{-5} + z^{-6} + z^{-7}$$

The time domain relationship between the input and the output is then

$$y[nT_s] = x[nT_s] + x[(n-1)T_s] + x[(n-2)T_s] + \ldots + x[(n-7)T_s]$$

Note that the output is simply a sum of the inputs over time, KT_s, as seen in Eq. (4.7). The time-domain impulse response of the first-order averaging filter is shown in Fig. 4.16. Note the rectangular shape. ■

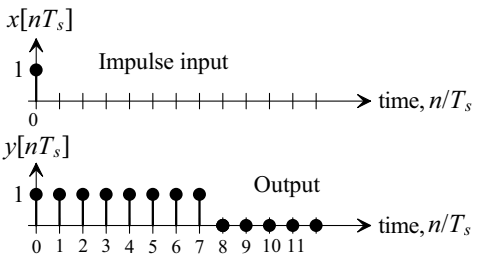


Figure 4.16 Impulse response of a K = 8 averaging filter.

Averaging without Decimation: A Review

The counter and the accumulate-and-dump discussed in Sec. 4.2.1 performed averaging and decimation in one stage. In other words, for example with $K = 4$, it summed four input samples, as shown in the sequence below, and passed the result to the output:

$$\overbrace{x(1)+x(2)+x(3)+x(4)}^{\text{First output}}+\ \overbrace{x(5)+x(6)+x(7)+x(8)}^{\text{Second output}}+x(9)+... \tag{4.16}$$

where the output clocking frequency is, as seen in Fig. 4.9, f_s/K. For the lowpass Sinc-response (averaging) filters discussed in this section

$$\begin{aligned}
&\overbrace{x(1)+x(2)+x(3)+x(4)}^{\text{First output}}+x(5)+x(6)+x(7)+x(8)+\ x(9)+...\\
&x(1)+\ \overbrace{x(2)+x(3)+x(4)+\ x(5)}^{\text{Second output}}+x(6)+x(7)+x(8)+\ x(9)+...\\
&x(1)+x(2)+\ \overbrace{x(3)+x(4)+\ x(5)+x(6)}^{\text{Third output}}+x(7)+x(8)+\ x(9)+...\\
&x(1)+x(2)+x(3)+\ \overbrace{x(4)+\ x(5)+x(6)+x(7)}^{\text{Fourth output}}+x(8)+\ x(9)+...
\end{aligned} \tag{4.17}$$

where the outputs of the averaging filter occur at the same rate as the inputs, f_s. The z-domain representation of the Sinc filter in Fig. 4.13 is the same as the counter or the accumulate-and-dump's transfer function

$$y(nT_s)=x[nT_s]+x[(n-1)T_s]+x[(n-2)T_s]+...\rightarrow Y(z)=X(z)(1+z^{-1}+z^{-2}+...+z^{1-K}) \tag{4.18}$$

or, again, a filter response given by Eq. (4.10)

Cascading Sinc Filters

The transfer function of a cascade of L of these Sinc-response filters can be written as

$$H(z)=\left[\frac{1-z^{-K}}{1-z^{-1}}\right]^L \tag{4.19}$$

$$|H(f)|=K^L\cdot\left[\frac{Sinc\left(K\pi\frac{f}{f_s}\right)}{Sinc\left(\pi\frac{f}{f_s}\right)}\right]^L \tag{4.20}$$

The attenuation through a cascade of L Sinc lowpass filters, see Eqs. (2.38) and (2.39) is

$$\left|\frac{\text{Main lobe}}{\text{First sidelobe}}\right|=K^L\cdot\sin^L\left(\frac{1.5\pi}{K}\right)\approx L\cdot 13\text{ dB for }K\geq 8 \tag{4.21}$$

while the droop at $f_s/2K$, see Fig. 4.17, is

$$\text{Droop}=\left[K\cdot\sin\left(\frac{\pi}{2K}\right)\right]^{-L}\approx L\cdot(-3.9)\text{ dB for }K\geq 8 \tag{4.22}$$

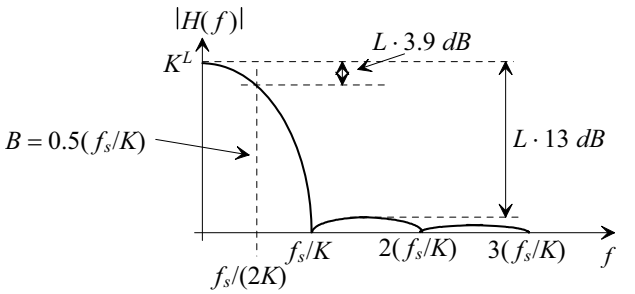


Figure 4.17 General frequency response of a lowpass Sinc (averaging) filter.

Example 4.5
Repeat Ex. 4.4 if an $L = 2$ averaging filter is used.

The transfer function of the filter is

$$H(z) = \left[\frac{1 - z^{-8}}{1 - z^{-1}}\right]^2 = 1 + 2z^{-1} + 3z^{-2} + 4z^{-3} + 5z^{-4} + ... + 3z^{-12} + 2z^{-13} + z^{-14}$$

The time domain relationship is

$$y[nT_s] = x[nT_s] + 2x[(n-1)T_s] + 3x[(n-2)T_s] + ... + 2x[(n-13)T_s] + x[(n-14)T_s]$$

The impulse response of the $L = 2$ lowpass Sinc filter is shown in Fig. 4.18. Note the triangular shape of the curve and how the impulse response of the $L = 2$ filter lasts twice as long as the $L = 1$ filter's response in Fig. 4.16. ■

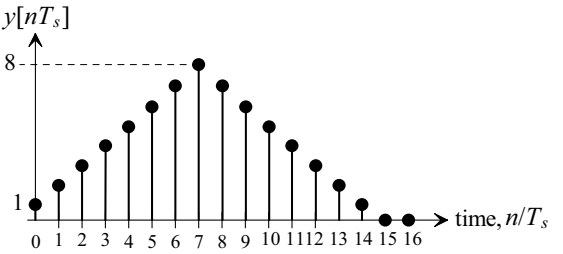


Figure 4.18 Impulse response of an L = 2 averaging filter with K = 8.

Finite and Infinite Impulse Response Filters

At this point a short note concerning digital filter names is in order. The comb filter seen in Fig. 1.24 (or Fig. 4.14), or the differentiator in Fig. 1.19, is an example of a *finite impulse response* (FIR) digital filter. Applying a unit amplitude impulse to the input of the comb filter causes the output of the comb filter to go to a 1 at the moment the impulse is applied and a −1 KT_s seconds later. The output is zero at all other times. In other words, the impulse response of the filter has a finite duration. The comb filter and differentiator are also called *non-recursive* filters since their outputs are only a function of their inputs (the output isn't fed back and used in the filter).

The integrator (also sometimes called an accumulator) shown in Figs. 1.26, 1.28, and 4.14 is an example of an *infinite impulse response* (IIR) digital filter. Applying a unit amplitude impulse to the input of the digital integrator, with zeroes the remaining times, causes the output of the integrator to increase to one and remain at one indefinitely. In other words, the output response of the integrator is of infinite duration. The integrator is also called a *recursive* filter since its current output value is a function of previous output values. Any digital filter with a denominator is a recursive filter.

While the averaging filters seen in Figs. 4.13 and 4.14 use an integrator (IIR section) and a comb filter (FIR section), overall, as seen in the previous two examples, it exhibits FIR behavior. Further, since the averaging filter's output is only a function of its inputs, as seen in Eq. (4.17), it is a non-recursive filter.

4.2.3 Bandpass and Highpass Sinc Filters

So far we've focused on lowpass filters. There are many situations, especially in communication systems, where we may want to filter or perform data conversion on a range of frequencies that doesn't extend from DC to B ($=f_s/2K$). Bandpass ADCs and DACs, for example, are popular in communication systems. In this section, we introduce bandpass and highpass sinc-shaped filters.

Canceling Zeroes to Create Highpass and Bandpass Filters

In Fig. 4.13 we saw that we can generate a lowpass filter by canceling the zero at DC in a comb filter. We can generate a highpass, or differencing, filter by canceling a comb filter zero at $f_s/2$, as seen in the example shown in Fig. 4.19 with $K = 8$. The same equations, Eqs. (4.21) and (4.22), can be used to describe the behavior of this filter where, in the highpass response, the main lobe has shifted to $f_s/2$. Also, when looking at Fig. 4.19, remember that the frequency response of a digital filter is periodic with period f_s.

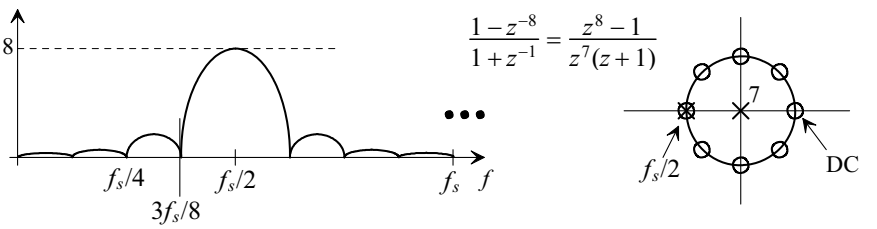


Figure 4.19 A highpass filter implementation using a comb filter.

We can generate a bandpass filter by canceling the zeroes at $f_s/4$ and $3f_s/4$, or some other frequencies, using a *digital resonator*. The general topology of the bandpass digital filter is shown in Fig. 4.20. Keeping in mind that the digital resonator is used to cancel the zeroes of the comb filter, we can write

$$H_D(z) = \frac{1}{1 - 2\cos\left[2\pi\frac{f}{f_s}\right]\cdot z^{-1} + z^{-2}} = \frac{z^2}{z^2 - 2\cos\left[2\pi\frac{f}{f_s}\right]\cdot z^1 + 1} = \frac{z^2}{\left(z - e^{+j\cdot 2\pi\frac{f}{f_s}}\right)\left(z - e^{-j\cdot 2\pi\frac{f}{f_s}}\right)} \tag{4.23}$$

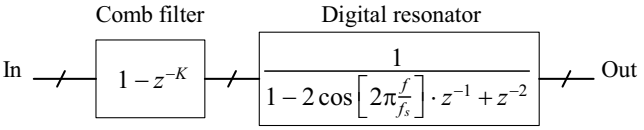


Figure 4.20 Implementing a sinc-shaped bandpass filter.

The time-domain representation of this equation is

$$y[nT_s] = 2\cos\left[2\pi\frac{f}{f_s}\right] \cdot y[(n-1)T_s] - y[(n-2)T_s] + x[nT_s] \tag{4.24}$$

It's desirable to determine at which frequencies the cosine term is an integer, zero, or a value that results in a trivial multiplication, that is, a shift so that we can implement the bandpass filter with trivial multiplications. In other words, we want a filter that uses only delays and additions so that its implementation is simple. The first frequency we will investigate is $f_s/4$. At this frequency the cosine term is zero and the digital-resonator/comb filter transfer function (the bandpass transfer function in Fig. 4.20) reduces to

$$H(z) = \frac{1 - z^{-K}}{1 + z^{-2}} \tag{4.25}$$

The magnitude and z-plane response of this filter, for $K = 8$, is shown in Fig. 4.21.

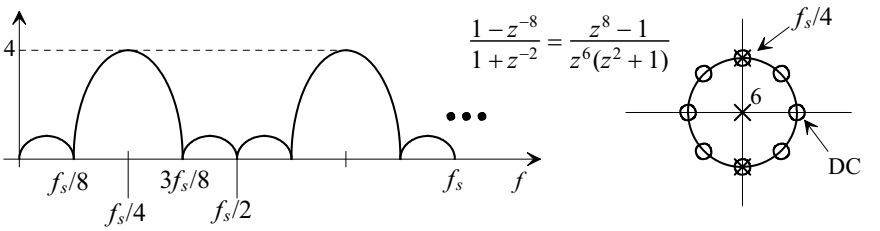


Figure 4.21 A bandpass filter implementation using a comb filter and digital resonator.

We can determine the magnitude response of Eq. (4.25) following the same procedure used to determine Eq. (2.38). The result, for the $f_s/4$ resonator, is

$$|H(f)| = \frac{\sqrt{2\left(1 - \cos K2\pi\frac{f}{f_s}\right)}}{\sqrt{2\left(1 + \cos 4\pi\frac{f}{f_s}\right)}} = \left|\frac{\sin\left(K\pi\frac{f}{f_s}\right)}{\cos\left(2\pi\frac{f}{f_s}\right)}\right| \tag{4.26}$$

At the center of the passband, that is $f_s/4$, $|H(f)| = K/2$. The ratio of the main lobe to the first side lobe, on either side, is plotted in Fig. 4.22 along with the lowpass Sinc filter response and is calculated using

$$\left|\frac{\text{Main lobe}}{\text{First side lobe}}\right| = \frac{K}{2} \cdot \sin\frac{3\pi}{K} \tag{4.27}$$

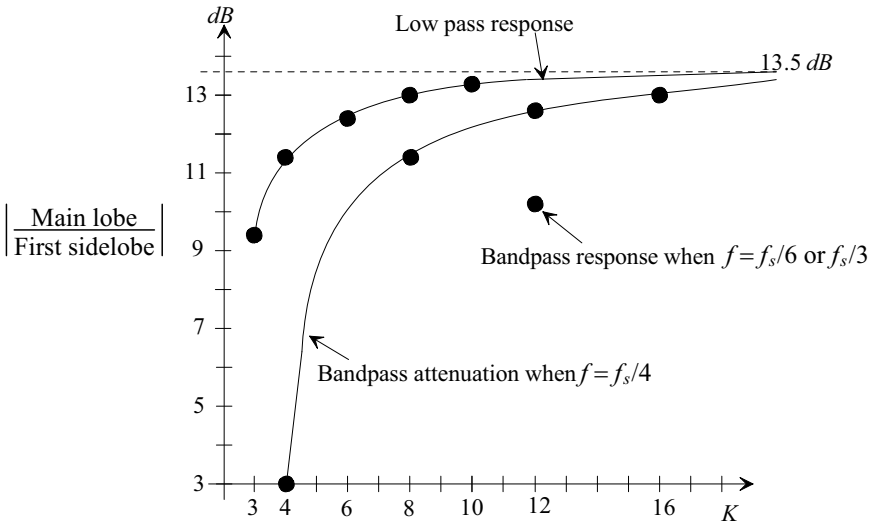


Figure 4.22 Lowpass and bandpass filter attenuation versus number of comb filter zeroes, K.

The cosine term in Eq. (4.24) can be set to ± 1 when $f = f_s/6$ or $f_s/3$ resulting in a bandpass filter. It should be clear that with the appropriate choice of sampling frequency, number of zeroes K used in the comb filter, and value of the cosine term, many different combinations of simple bandpass filters can be implemented using these techniques. There are, however, only a few resonators that can be implemented using binary numbers. The resonators' coefficients must be such that the poles perfectly lie on the unit circle and cancel the comb filter's zeroes on the unit circle.

The ratio of the main lobe to the first sidelobe for $f = f_s/6$ or $f_s/3$ is given, assuming $K = 12, 24, \ldots$, by

$$\left|\frac{\text{Main lobe}}{\text{First side lobe}}\right| = \frac{K \sin\left(\frac{3\pi}{2K}\right) \sin\left(\frac{\pi}{3} - \frac{3\pi}{2K}\right)}{\sin\frac{\pi}{3}} = 1.15K \sin\left(\frac{3\pi}{2K}\right) \sin\left(\frac{\pi}{3} - \frac{3\pi}{2K}\right) \quad (4.28)$$

which is approximately 13.5 dB for $K = 24, 36, 48 \ldots$ and 10.15 dB for $K = 12$, Fig. 4.22.

In order to increase the amount of attenuation between the main lobe and the first side lobe in a bandpass filter implementation, we can cascade filter sections (as we did in the lowpass filter implementations discussed earlier). For example, cascading five $f_s/4$ bandpass filters with $K = 8$ will result in an attenuation of 57 dB. Also, note that by changing the sampling, or filter clock frequency f_s, we can easily change the bandpass filter's center frequency. A change in the clock frequency, and its selection, can easily be implemented using a counter and some control logic.

Example 4.6

Sketch the block level circuit diagram for an $f_s/4$ digital resonator.

From Eq. (4.24) the time domain representation of the $f_s/4$ resonator can be written as

$$y[nT_s] = x[nT_s] - y[(n-2)T_s]$$

The implementation is shown in Fig. 4.23. ■

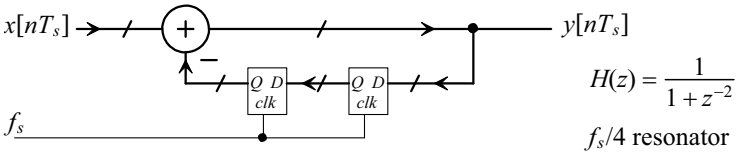


Figure 4.23 Implementation of a digital resonator.

Example 4.7

Assuming an 8-bit input word, sketch the implementation of the Sinc-shaped bandpass filter centered at $f_s/4$

$$\frac{1-z^{-8}}{1+z^{-2}}$$

What is the output word size? Using SPICE verify that the filter's frequency response is given by Eq. (4.26) with $K = 8$. Assume $f_s = 100$ MHz.

The sketch of the filter is seen in Fig. 4.24 (note the similarity to Fig. 4.14). The word size grows by $\log_2 K - 1$ from the input to the output. Here, where K is 8, the word grows 2-bits, to an output word size of 10-bits, (1-bit in the comb filter and 1-bit in the resonator for a gain of 4, see Fig. 4.21). Figure 4.25 shows the output of the resonator when the input is 25 MHz ($= f_s/4$). It can be useful to vary the input frequency and verify that Eq. (4.26) is correct. For example, if we change the input frequency to 12.5 MHz the filter's output is zero (this is the first adjacent zero point in the frequency response). When the input frequency is 15 MHz, the output of the filter is

$$|H(f)| = \left|\frac{\sin\left(K\pi\frac{f}{f_s}\right)}{\cos\left(2\pi\frac{f}{f_s}\right)}\right| = \left|\frac{\sin\left(8\pi\frac{15}{100}\right)}{\cos\left(2\pi\frac{15}{100}\right)}\right| = \left|\frac{-0.588}{0.588}\right| = 1 \rightarrow 0.25 \text{ (scaled by 4)}$$

where, since we normalize the filter's output so that passband gain is 1 (the input equals the output), the output of the filter is 0.25 the input. In the next chapter we'll see that we can throw away some of the lower bits in the output word since they don't contribute to an increase in the SNR. ■

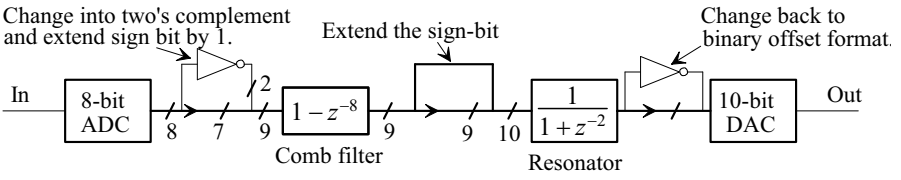


Figure 4.24 Digital filter sketch for Ex. 4.7.

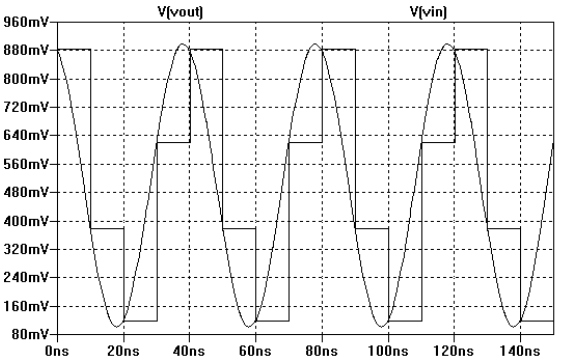


Figure 4.25 Input and output of the filter in Ex. 4.7 at $f_s/4$.

Frequency Sampling Filters

Consider the topology of a comb filter and resonators shown in Fig. 4.26. We are feeding the output of the comb filter through the resonators (with different center frequencies) and then using the combined sum of the resulting bandpass filter responses (the Sinc shapes) to build a bandpass filter. This is exactly the same as reconstructing a waveform in the time domain using an ideal RCF, as discussed in Ch. 2, except now we are using the

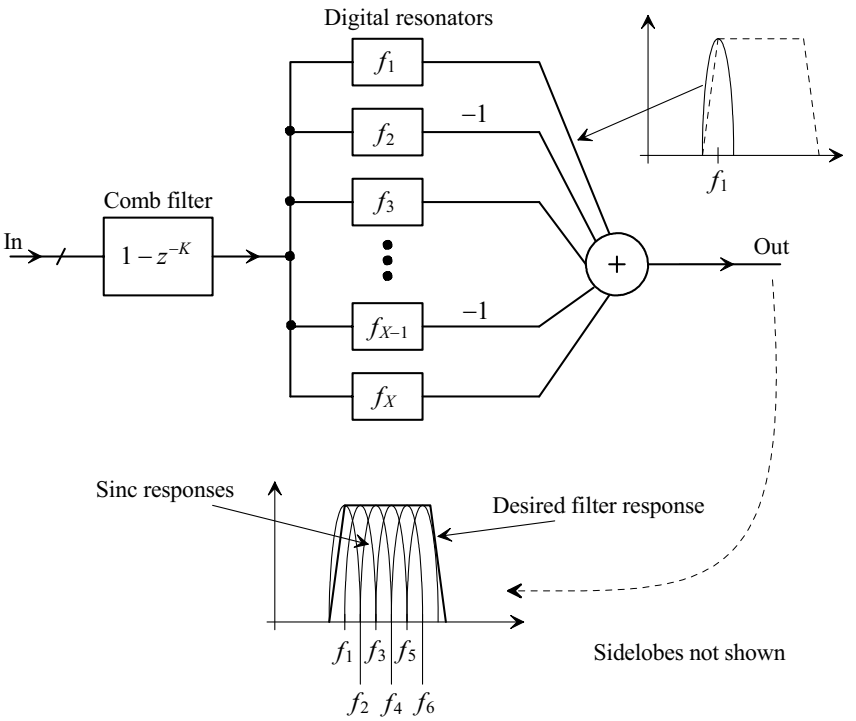


Figure 4.26 A frequency sampling filter.

summation of the frequency domain Sinc responses to generate a bandpass filter with a variable width. Note that every other digital resonator is subtracted rather than added to the final result. This is to account for the phase reversal between adjacent resonator outputs.

4.2.4 Interpolation using Sinc Filters

We saw back in Sec. 2.1.5 that an interpolator is a circuit that increases the clocking rate of the digital data. This was useful for, among other things, reducing the requirements placed on the reconstruction filter. The simplest interpolator was the input hold register, Fig. 2.28. Before we discuss interpolation using Sinc filters let's answer the question "Why can't we use an input hold register for interpolation here?"

Examine the block diagram in Fig. 4.27. The ADC is clocked at a rate of f_s so the anti-aliasing filter (AAF) limits the ADC's input spectrum to $f_s/2$. On the input of the ADC is a sample-and-hold (S/H) with a Sinc-shaped spectral response (–3.9 dB attenuation at the Nyquist frequency of $f_s/2$, see Fig. 2.17). We might expect, based on Fig. 2.29 and the associated discussion, that the hold register seen in Fig. 4.27 would provide an additional Sinc-shaped response in the signal path. However, remember (see Ex. 2.2) repetitively sampling and holding a signal results in only one S/H attenuation. Since the input signal in Fig. 4.27 sees this response when it passes through the S/H it isn't affected by another Sinc-shaped response as it passes through the hold register.

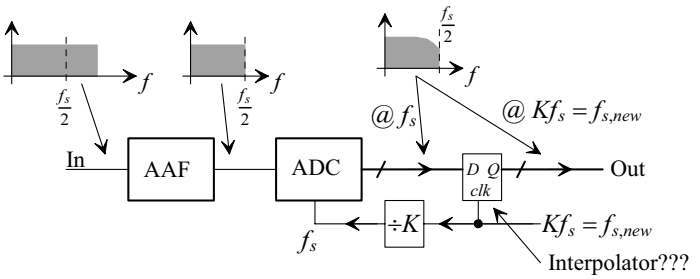


Figure 4.27 Interpolation using a hold register (see Sec. 2.1.5).

Note that because of this assumption that the data has passed through a S/H and experienced the associated Sinc-shaped filtering, we won't use zeroes padding in the discussions in this section. It should be straightforward to extend the discussions to designs that do use zeroes padding. Also note that if the digital data hasn't passed through a Sinc-response filter then passing it through a hold register, as discussed in Sec. 2.1.5, does introduce a Sinc-response (which, as we'll find out now, may often be desirable when performing interpolation).

The next question is "Why would we want the signal to see additional filtering?" The answer to this comes from reviewing Fig. 2.26. The image removal filter is an important component of an interpolator. We can implement this filter using

$$H(z) = \left[\frac{1 - z^{-K}}{1 - z^{-1}}\right]^L \tag{4.29}$$

The benefit, for an interpolator, comes from realizing that if we clock the comb filter with a slower clock, f_s/K, then one clock delay is KT_s so we can simplify the comb sections using

$$1 - z^{-K} \rightarrow 1 - z^{-1} \tag{4.30}$$

Figure 4.28 shows an interpolator implemented using a cascade of Sinc Filters. The attenuation through the interpolator is given, see Eq. (2.37), by

$$|H(f)| = \left| \frac{Sinc\left(K\pi \frac{f}{f_{s,new}}\right)}{Sinc\left(\pi \frac{f}{f_{s,new}}\right)} \right|^L \tag{4.31}$$

Larger attenuation can be achieved by cascading more stages, L. Again, the word size grows by 1-bit through each comb filter and by $\log_2 K - 1$ through each integrator. Also note that if we were to use zeroes padding instead of the inherent hold register present in the topology (see discussion on previous page) a selector, see Fig. 2.26, would be added in between the comb filters and integrators to introduce $K - 1$ zeroes.

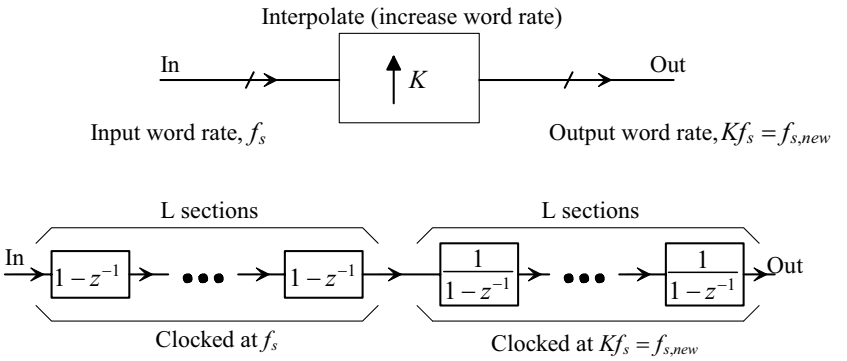


Figure 4.28 Interpolating using Sinc filters for image removal.

Example 4.8

Using an 8-bit word, an input clock rate of 12.5 MHz, an input frequency of 1.1 MHz, and K = 8 determine the output word size, magnitude, and output clock rate for interpolators with L = 1 and 2. Also determine the attenuation between the passband and the first sidelobe, see Fig. 2.29. Verify your results with SPICE.

The Nyquist frequency is 6.25 MHz (the spectral content on the input of the ADC should be < 6.25 MHz to avoid aliasing). The output clock rate is 100 MHz. For the case when L = 1 the word grows 1-bit through the comb filter and 2-bits through the integrator so the output word size is 11-bits. When L = 2 the output word size is 14 bits. We'll see in the next chapter that the signal-to-noise ratio sets the number of bits we keep on the output of the filter or in between adjacent stages.

The attenuation through the filter is

$$|H(f)| = \left| \frac{Sinc\left(K\pi \frac{f}{f_{s,new}}\right)}{Sinc\left(\pi \frac{f}{f_{s,new}}\right)} \right|^L = \left| \frac{Sinc\left(8\pi \frac{1.1}{100}\right)}{Sinc\left(\pi \frac{1.1}{100}\right)} \right|^L \approx \left| \frac{\sin\left(8\pi \frac{1.1}{100}\right)}{8\pi \frac{1.1}{100}} \right|^L = (0.988)^L$$

or, for L = 1, – 0.107 dB and for L = 2, – 0.214 dB. We can estimate the attenuation between the passband and first sidelobe using information seen in Fig. 2.30. For L = 1 the attenuation is roughly 13 dB and for L = 2 the attenuation is 26 dB.

Figure 4.29 shows the simulation results. Looking closely we see that for L = 1 the interpolator's output is simply a linear change between adjacent output points. For L = 2 we see additional phase shift (more delay through the filter) and a more "rounded" behavior indicating a better representation of the original signal. It can be instructive to vary the input frequency in these simulations and look at the resulting output signals (especially showing how the output signal amplitude drops as the input signal moves closer to the Nyquist frequency of 6.25 MHz). ■

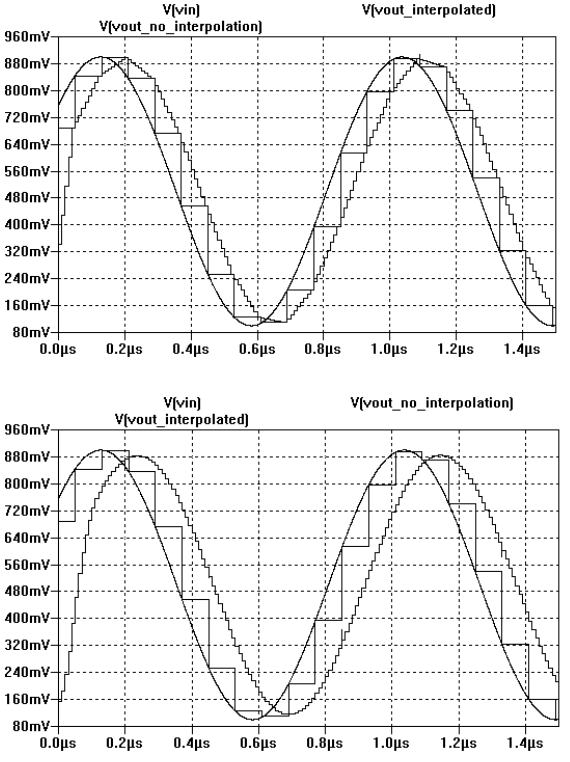


Figure 4.29 Interpolation using one- and two-stage Sinc filters.

Additional Control

In order to have an additional parameter for adjusting the image removal filter characteristics, Eqs. (4.29) and (4.31), consider adding delay to our comb filter sections as seen in Fig. 4.30.

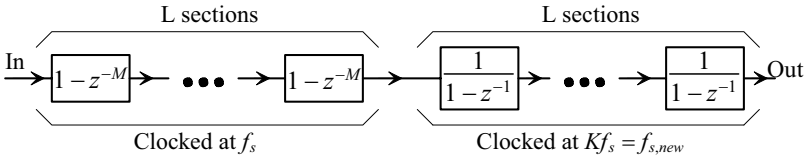


Figure 4.30 General interpolation using Sinc filters.

Using this topology we can describe the filter's characteristics using

$$H(z) = \left[\frac{1-z^{-KM}}{1-z^{-1}}\right]^L \tag{4.32}$$

and

$$|H(f)| = \left|\frac{Sinc\left(KM \cdot \pi \frac{f}{f_{s,new}}\right)}{Sinc\left(\pi \frac{f}{f_{s,new}}\right)}\right|^L \tag{4.33}$$

Figure 4.31 plots this equation. Note that the Nyquist frequency (= $f_s/2$) remains unchanged since it's set by the sampling in the ADC. By using the factor M we can achieve a narrower bandwidth but at the cost of more droop at the Nyquist frequency. The word size, in this filter, still grows by 1-bit in every comb section and by $\log_2 K - 1$ bits in the integrator sections.

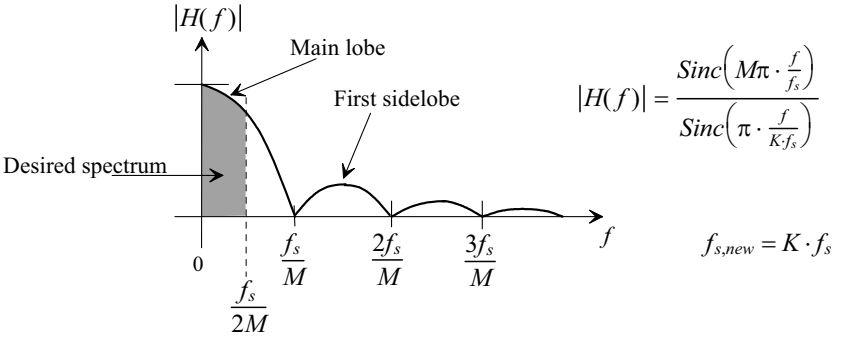


Figure 4.31 Frequency response image removal filter using a Sinc interpolator, Fig. 4.30.

Cascade of Integrators and Combs

A quick note before leaving this section. The filters we presented are often labeled CIC filters since they are formed by cascading integrators and comb filters. We'll continue to avoid using this term for reasons that will become clear in the next section.

4.2.5 Decimation using Sinc Filters

As seen in Fig. 2.12, resampling digital data at a lower frequency (decimation) consists of passing the data through a digital anti-aliasing filter followed by resampling. In this section we discuss using a Sinc filter, Eqs. (4.10) and (4.12), with a response seen in Fig. 4.10. The output word rate of the decimator is f_s/K and the input desired spectrum is limited to DC to $f_s/2K$ (as seen in Fig. 4.10).

Notice that, in the filters we've covered so far in this chapter, we've put the comb filter or differentiator before the integrators or resonators. This was to eliminate unwanted overflow problems in the latter (which are recursive filters). After reviewing Fig. 4.28 we might try putting the integrators before the comb filters. While this may work for small signals, or by using very wide registers in the integrators, it would be better to avoid overflow problems altogether. In order to move towards this goal lets equate Eq. (4.8) and (4.29)

$$\begin{aligned} H(z) &= \left[\frac{1-z^{-K}}{1-z^{-1}}\right]^L = \left[\sum_{n=0}^{K-1} z^{-n}\right]^L = [1+z^{-1}+z^{-2}+\ldots+z^{1-K}]^L \\ &= \left[(1+z^{-1})\cdot(1+z^{-2})\cdot\ldots\cdot(1+z^{-2^{\log_2 K-1}})\right]^L \\ &= (1+z^{-1})^L\cdot(1+z^{-2})^L\cdot\ldots\cdot(1+z^{-2^{\log_2 K-1}})^L \end{aligned} \tag{4.34}$$

or we can implement decimation using only non-recursive averagers (see Fig. 1.16 and the associated discussion). The word size increases by 1-bit through each averager. Note that we are assuming the decimation is a factor of 2 here. Figure 4.32 shows the implementation of the decimator. The registers are used to re-sample the data. The higher factors of delay are implemented by clocking the registers with the slower clock (clocking an averager, $1+z^{-1}$, with a clock of f_s/K implements $1+z^{-K}$). This topology is called, by some authors, a CIC filter (see comment at the bottom of previous page) even though there aren't any integrators present in the topology. Figure 4.33 shows example spectrums when using this decimator. Note that the sidelobes alias into the desired spectrum. By increasing the order of the filter L, we can reduce the effects of aliasing.

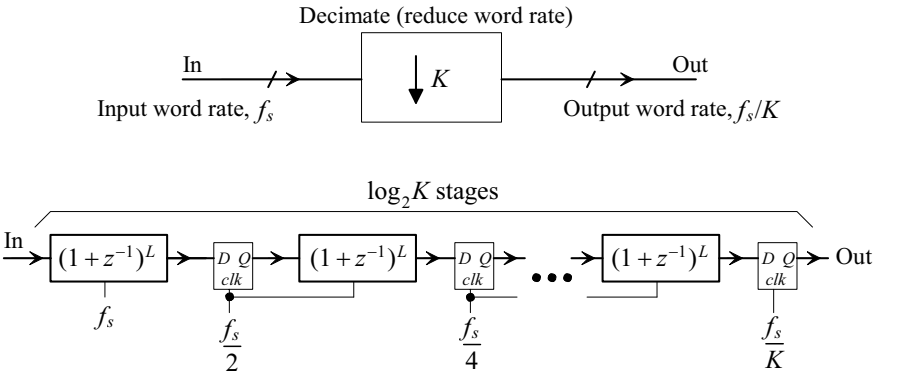


Figure 4.32 Decimation using Sinc anti-aliasing filters.

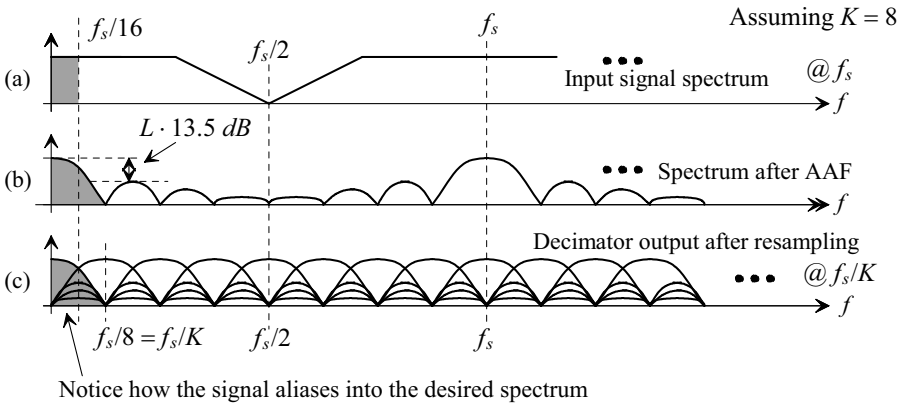


Figure 4.33 Spectrums when decimating and a Sinc anti-aliasing filter used.

Example 4.8

Using an 8-bit word, an input clock rate of 100 MHz, an input frequency of 1.1 MHz, and $K = 8$ determine the output word size, magnitude, and output clock rate for Sinc-decimators with $L = 1$ and 2. Verify your results with SPICE.

The transfer function of the anti-aliasing filter inherent in the decimator is given by Eq. (4.34) with $K = 8$ and $L = 1$ or 2. The output word rate is 12.5 MHz. The simulation results are seen in Fig. 4.34. Note that the decimated output with $L = 1$ looks essentially the same as the output with $L = 2$ (except for more delay when $L = 2$). This (the shape of the outputs being more or less the same) wasn't the case for the interpolator outputs seen in Fig. 4.29. For the interpolator, where aliasing isn't a concern, increasing L results in a reduction in the aliased signals present on the output of the interpolator. Increasing L allows the interpolator output to move closer to the original input (in the extreme, $L \to \infty$, and we get the original analog waveform). By increasing the order, L, in the decimation filter we reduce the aliased signals present in our desired spectrum, Fig. 4.34. ■

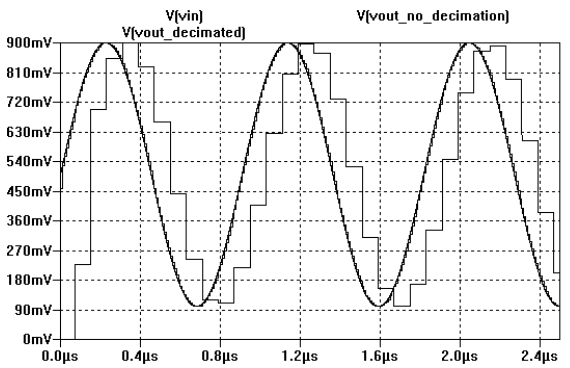


Figure 4.34 Decimating by 8, an example.

4.3 Filtering Topologies

The Sinc filters discussed in the last section are very useful for general mixed-signal circuit design, especially when interpolating or decimating, since they don't employ complicated multiplications. Unfortunately, they also don't have sharp filtering characteristics and there isn't a lot of flexibility when selecting the filter's frequency response. In this section we'll present some additional filtering topologies mostly based on the integrator. Most of these topologies are directly related to topologies discussed in the last chapter covering analog filters.

4.3.1 FIR Filters

Towards understanding the reason for this approach, consider the non-recursive FIR filter topology seen in Fig. 4.35. The transfer function of the filter is

$$H(z) = A_0 + A_1 z^{-1} + A_2 z^{-2} + A_3 z^{-3} \tag{4.35}$$

If, for example, we set all of the filter's coefficients to 1 then we can write, as seen in Eqs. (4.9) and (4.10),

$$H(z) = \frac{1 - z^{-4}}{1 - z^{-1}} \tag{4.36}$$

or, again, a Sinc-shaped lowpass comb filter. When compared to the comb filters used earlier, Figure 1.24, this topology has more adders.

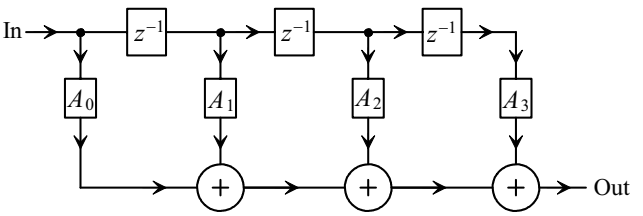


Figure 4.35 A four-stage FIR filter.

As another example consider setting all of the coefficients to 0.25. This is done by shifting the word left two bits or by extending the sign-bit by two bits. Extending the sign bit increases the word size and ensures we don't lose resolution (but results in more hardware). Figure 4.36 shows the resulting filter's step response. Note the similarity to a first-order RC circuits step response. Also note that the impulse response for this filter simply reveals the filter's coefficients (here all 0.25). Based on a desired step response a heuristic approach can be used to design the filter.

The benefits of using FIR filters are that they are inherently stable (they are non-recursive so no feedback is used) and they can have linear phase (constant delay). The drawback is that they, for a given number of delays, aren't as good at filtering as the recursive structures we'll talk about in the rest of this section. Unfortunately, recursive structures are subject to instability. In addition, topologies using integrators are subject to overflow. Let's discuss these two issues before going any further.

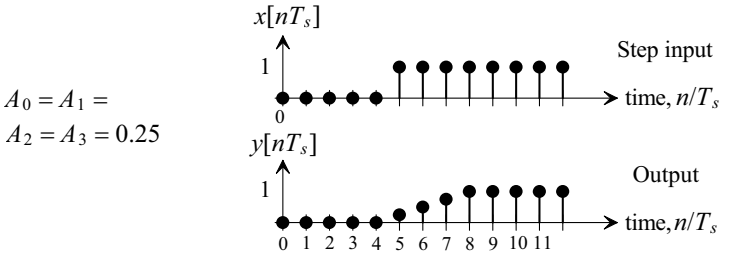


Figure 4.36 Step response of a 4-stage FIR filter with all coefficients set to 0.25.

4.3.2 Stability and Overflow

A weighted integrating filter is seen in Fig. 4.37. The output of the circuit is fed back to the input after it is multiplied by a. The output of the circuit in the time-domain may be written as

$$y[nT_s] = x[(n-1)T_s] + a \cdot y[(n-1)T_s] \tag{4.37}$$

or

$$y[nT_s] = x[(n-1)T_s] + a \cdot x[(n-2)T_s] + a^2 \cdot x[(n-3)T_s] + a^3 \cdot x[(n-4)T_s] + \ldots, \tag{4.38}$$

which will obviously blow up if $a > 1$.

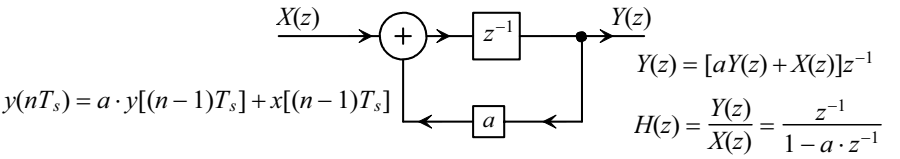


Figure 4.37 A weighted integrating filter.

The z-domain representation of Eq. (4.37) is

$$H(z) = \frac{1}{z - a} \tag{4.39}$$

Figure 4.38 shows the z-plane and magnitude plots for this equation. If $a > 1$ then $H(z)$ becomes unstable. So *for a stable system* we must require our poles to reside within the unit circle. (There are no restrictions on the location of zeroes.) This sounds simple enough; however, notice that we have, in most of the previously discussed digital filters, placed poles right on the unit circle. If there is rounding in our digital numbers, we could be faced with an unstable digital filter. This would be a very common occurrence in a digital filter implemented using software, if care was not taken to avoid rounding errors. Since we use integer numbers in our hardware implementations, instability shouldn't be a problem unless we start to try to round numbers to decrease hardware complexity (performing divisions or multiplications) without being careful.

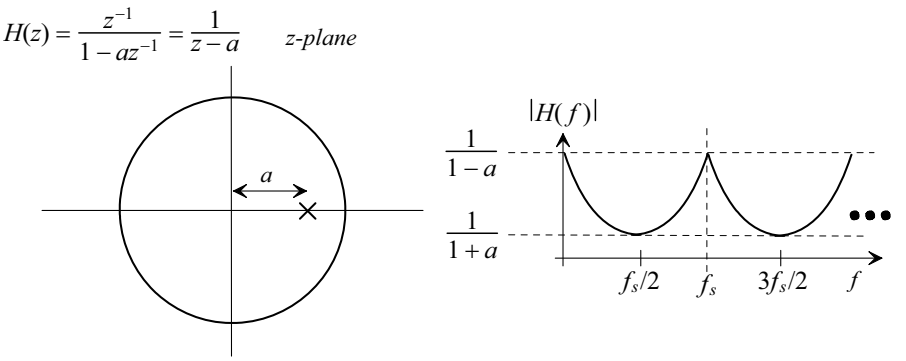
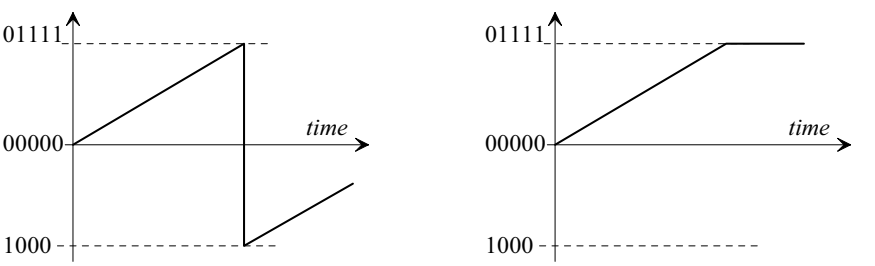


Figure 4.38 The z-plane representation and magnitude response for a weighted integrating filter.

Overflow

Recall that when we used integrators with comb filters earlier in the chapter we increased the input/output word size of the integrators by $\log_2 K - 1$ while the comb filter outputs increased by 1-bit for an overall increase of $\log_2 K$. We calculated these values by determining the register size required to sum K words. In a recursive filter the output is fed back and summed at various points in the filter, so determining the exact register size required can be challenging. Figure 4.39a shows how an integrator overflows causing the output to wrap around (go from the most positive value to the most negative value or vice-versa). In Fig. 4.39b we show the more desirable situation where the output saturates. This keeps the filter from becoming unstable; however, it will introduce nonlinearities in the filter's response. Nonlinearity at some extreme is usually better than instability. The question is, "How do we determine overflow?" We know that we can't look at the carry out bit because, as we saw in Figs. 4.7 and 4.8, the adder overflowing is a normal occurrence when adding two's complement numbers. What we do know is that if the input sign bits are both 0 then the output sign bit must be 0 (if both inputs are positive then the output must be positive). The same can be said for adding negative numbers. Figure 4.40 shows how we can modify an integrator to avoid overflow.



(a) Integrator overflow

(b) Clamping the output when overflow occurs.

Figure 4.39 Integrator overflow and clamping the integrator's output.

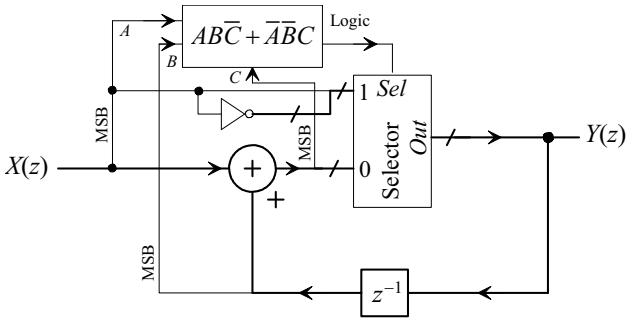


Figure 4.40 Modifying the integrator to avoid overflow.

4.3.3 The Bilinear Transfer Function

The bilinear transfer function and its implementation using an integrator and differentiator were presented back in Sec. 3.2.1. Figure 4.41 shows the digital implementation of the bilinear filter seen in Fig. 3.29. It's important, at this point, to see how the continuous-time implementation in Fig. 3.29 is directly implemented in Fig. 4.41. In particular we note that

$$G_1 = G_{1D} \cdot f_s \text{ and } G_3 = \frac{G_{3D}}{f_s} \tag{4.40}$$

and so

$$\frac{v_{out}(f)}{v_{in}(f)} = \frac{1}{G_2} \cdot \frac{1+\frac{s}{1/G_3}}{1+\frac{s}{G_1 G_2}} = \frac{1}{G_2} \cdot \frac{1+j \cdot \frac{f}{f_s/(2\pi G_{3D})}}{1+j \cdot \frac{f}{f_s G_{1D} G_2/2\pi}} \tag{4.41}$$

The location of the pole is given by

$$f_{3dB,pole} = \frac{f_s G_{1D} G_2}{2\pi} \tag{4.42}$$

while the location of the zero is at

$$f_{3dB,zero} = \frac{f_s}{2\pi G_{3D}} \tag{4.43}$$

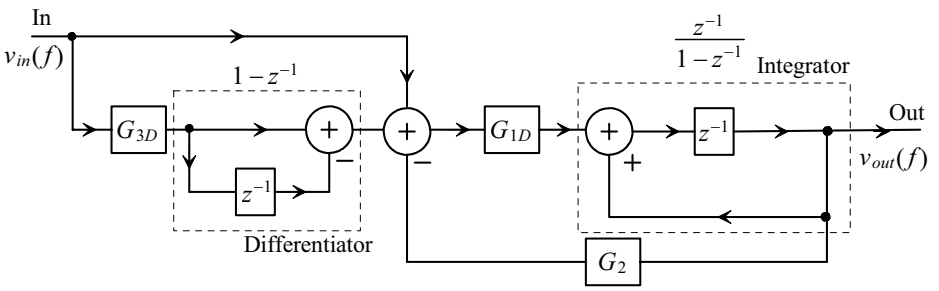


Figure 4.41 Digital implementation of the bilinear transfer function.

Note that in order for our filter to be useful the frequencies of interest must be much lower than the filter's clocking frequency, f_s. In other words, the frequencies where the pole and zero are located must be much less than f_s. This means, assuming $G_2 = 1$, that $G_{1D} << 1$ and $G_{3D} >> 1$.

Let's attempt to simplify this filter. If we look at the transfer function, Eq. (4.41), we see that the feedback gain, G_2, simply scales the amplitude of the transfer function and can be used to further adjust the location of the filter's pole. Because we can scale the amplitude of the signal either before or after the filter, and independent of the filter's operation, and we can precisely set the pole of the filter using G_{1D}, we can, without loss of functionality, set G_2 to 1. We can then rearrange the summing, delaying, and multiplying blocks, as seen in Fig. 4.43. Using these results we can write the z-domain representation of the transfer function, Eq. (4.41), as

$$\frac{v_{out}(z)}{v_{in}(z)} = \frac{G_{1D}(1+G_{3D})}{z-(1-G_{1D})} \cdot \frac{z - G_{3D}/(1+G_{3D})}{z} \tag{4.44}$$

Before attempting to simplify the filter implementation seen in Fig. 4.43 further, let's show that, indeed, Eq. (4.44) is equivalent to Eq. (4.41) when $f << f_s$. It will be helpful to remember that

$$z \approx 1 + \frac{s}{f_s} \text{ and } z^{-1} \approx 1 - \frac{s}{f_s} \text{ if } f << f_s \text{ or } z \approx 1 + \frac{s}{f_s} \approx \frac{1}{1-\frac{s}{f_s}} \approx \frac{1}{z^{-1}} \text{ if } \frac{s^2}{f_s^2} \approx 0 \tag{4.45}$$

where $s = j\omega = j2\pi f$. Rewriting Eq. (4.44) in the s-domain results in

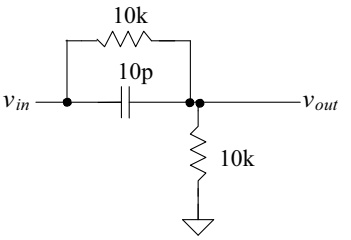
$$\frac{v_{out}(f)}{v_{in}(f)} = \frac{G_{1D}(1+G_{3D})}{1+\frac{s}{f_s}-1+G_{1D}} \cdot \left[1 - \left(1 - \frac{s}{f_s}\right) G_{3D}/(1+G_{3D})\right] \tag{4.46}$$

$$= \frac{1+\frac{s}{f_s/G_{3D}}}{1+\frac{s}{G_{1D}\cdot f_s}} = \frac{1+j\cdot\frac{f}{f_s/(2\pi G_{3D})}}{1+j\cdot\frac{f}{G_{1D}\cdot f_s/2\pi}} \tag{4.47}$$

which is clearly the same as Eq. (4.41) when $G_2 = 1$.

Example 4.9

Sketch, and determine the transfer function for, the digital filter equivalent of the following RC circuit. Assume the digital filter is clocked at 100 MHz.



$$\frac{v_{out}(f)}{v_{in}(f)} = \frac{1}{2} \cdot \frac{1 + j\omega \cdot 100\,\text{ns}}{1 + j\omega \cdot 50\,\text{ns}}$$

Figure 4.42 A simple first-order RC circuit.

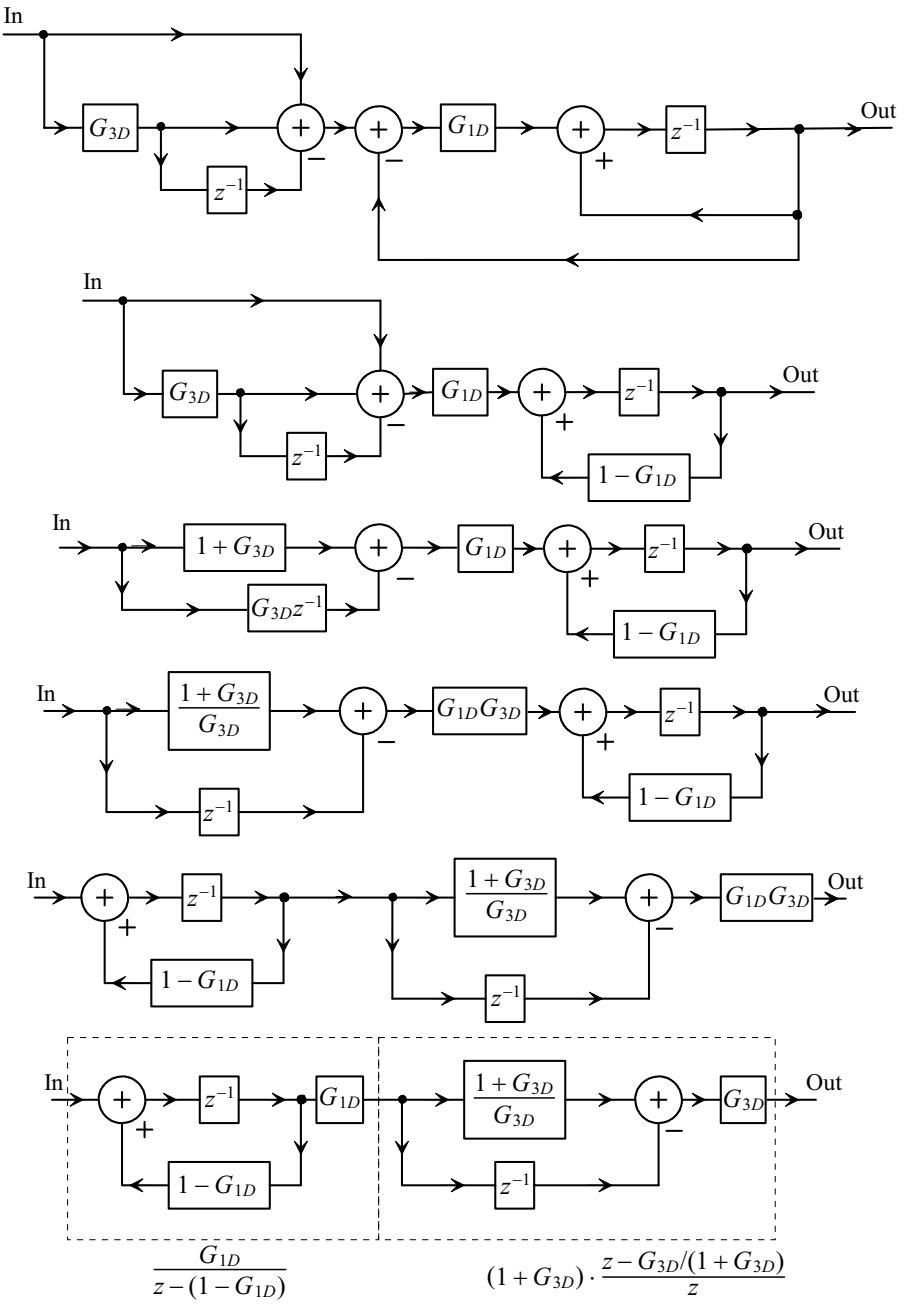
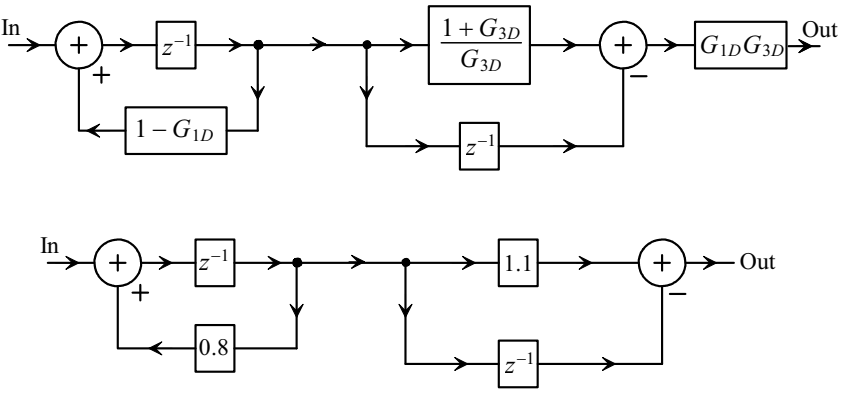


Figure 4.43 Simplifying the digital implementation of the bilinear filter.

Comparing the transfer function in Fig. 4.42 to Eq. (4.47), we see that if

$$\frac{G_{3D}}{f_s} = 100\ ns \rightarrow G_{3D} = 10 \text{ and } \frac{1}{G_{1D} \cdot f_s} = 50\ ns \rightarrow G_{1D} = 0.2$$

then we can use any of the filters in Fig. 4.43. The sketch of the digital filter is seen in Fig. 4.44. The multiplication of the transfer function by 1/2 is nulled by the multiplication by $G_{1D}G_{3D}\,(=2)$. To verify that the filter in Fig. 4.44 functions as desired at DC, we see that the output of the first stage is 0.5 when the input is 0.1, and the output of the second stage is 0.05 (with a 0.5 on its input). ■



$$H(z) = 1.1 \cdot \frac{z - 10/11}{z(z - 0.8)} = 1.1 \cdot \frac{z^{-1}}{1 - 0.8z^{-1}} \cdot (1 - z^{-1} \cdot 10/11)$$

Figure 4.44 Digital filter from Ex. 4.9.

The Canonic Form (or Standard Form) of a Digital Filter

Studying Fig. 4.44, we might wonder if we can further reduce the size of the digital filter. We see in this figure that it may be possible to eliminate the second delay element (which, of course, is a register) and use only a single delay. The result of this modification is seen in Fig. 4.45. Intuitively we would think that the phase response of the filter will change because, now, there is less delay in series with input of the second adder. We can write the output as

$$\frac{v_{out}(z)}{v_{in}(z)} = G_{1D}G_{3D}\left[\frac{1 + G_{3D}}{G_{3D}} \cdot \frac{1}{1 - z^{-1}(1 - G_{1D})} - \frac{z^{-1}}{1 - z^{-1}(1 - G_{1D})}\right] \quad (4.48)$$

or

$$\frac{v_{out}(z)}{v_{in}(z)} = G_{1D}(1 + G_{3D}) \cdot \frac{z - G_{3D}/(1 + G_{3D})}{z - (1 - G_{1D})} \quad (4.49)$$

which is clearly the same response as derived in Fig. 4.43 or Eq. (4.44) except that the output is one clock cycle, z, earlier. This reduced delay has no effect on the magnitude response of the filter and little effect, assuming $f << f_s$, on the phase response of the filter.

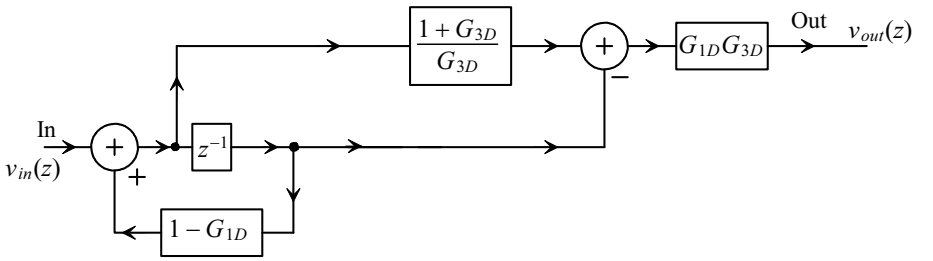


Figure 4.45 Canonic form of a first-order digital filter.

The general form of this first-order canonic (or standard form) filter is seen in Fig. 4.46. The filter is termed *canonic* because the minimum number of delays are used. One delay is used for each pole. Remember that in order for a digital filter to be realizable in hardware there must be fewer or an equal number of zeroes than poles in a filter's transfer function.

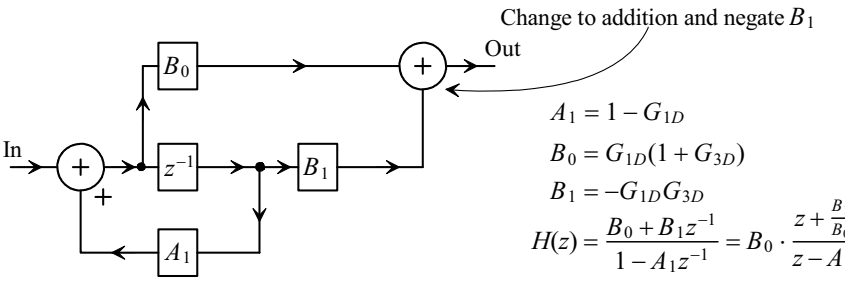


Figure 4.46 General canonic form of a first-order digital filter.

We can, again, derive the transfer function for the first-order bilinear digital filter. This time, however, let's use the variables in Fig. 4.46. Again, assuming $f << f_s$, and using Eq. (4.45) results in

$$H(f) = \frac{B_0 + B_1}{1 - A_1} \cdot \frac{1 + \frac{s}{f_s\left(1+\frac{B_1}{B_0}\right)}}{1 + \frac{s}{f_s(1-A_1)}} \tag{4.50}$$

where

$$f_{3dB,pole} = \frac{f_s(1 - A_1)}{2\pi} \tag{4.51}$$

and

$$f_{3dB,zero} = \frac{f_s}{2\pi}\left(1 + \frac{B_1}{B_0}\right) \tag{4.52}$$

and the gain at DC is

$$A_{DC} = \frac{B_0 + B_1}{1 - A_1} \tag{4.53}$$

Example 4.10
Using the canonic form of the first-order digital filter, repeat Ex. 4.9.

Comparing Eq. (4.50) with the transfer function in Fig. 4.42, we can write

$$2\pi f_{3dB,pole} = \frac{1}{50\text{ ns}} = f_s \cdot (1 - A_1) = 100\,\text{MHz}(1 - A_1) \rightarrow A_1 = 0.8$$

$$A_{DC} = \frac{1}{2} = \frac{B_0 + B_1}{1 - A_1} = \frac{B_0 + B_1}{1 - 0.8} \rightarrow B_0 + B_1 = 0.1$$

$$2\pi f_{3dB,zero} = \frac{1}{100\text{ ns}} = f_s \cdot \left(1 + \frac{B_1}{B_0}\right) = 100\,\text{MHz}\left(\frac{0.1}{B_0}\right) \rightarrow B_0 = 1$$

and thus $B_1 = -0.9$. The filter's sketch is seen in Fig. 4.47. We will discuss how to implement the multipliers later.

Let's do a quick check to see if the filter functions as desired at DC. If we apply 0.1 to the input of the filter then, according to the transfer function in Fig. 4.42, the output of the filter should be 0.05 or one-half the input. Because the input to the filter is a DC signal, both sides of the delay will have the same value. According to Fig. 4.38, this value will be 0.5 (the output of the weighted integrator is 1/[1 − 0.8] times the input signal, here 0.1, at DC). The output will then be 0.5 – 0.45 or 0.05 (as we would expect). ■

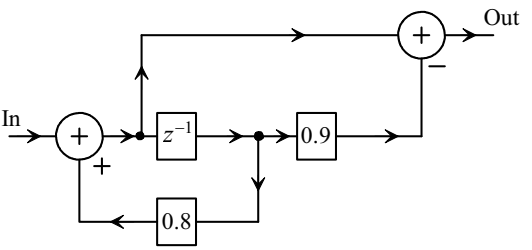


Figure 4.47 Canonic form of the first-order digital filter in Ex. 4.10.

Example 4.11
Sketch the digital filter implementation of the lowpass filter in Ex. 3.1 from the last chapter that has a DC gain of 1 and a 3 dB frequency of 1.59 MHz. Assume the filter is clocked at 100 MHz.

The filter's continuous-time frequency response is given by

$$H(f) = \frac{1}{1 + j \cdot \frac{f}{1.59\text{ MHz}}}$$

Using Eqs. (4.50) to (4.53), we begin by calculating A_1

$$f_{3dB,pole} = 1.59\text{ MHz} = \frac{f_s(1-A_1)}{2\pi} = \frac{100\text{ MHz}}{2\pi}(1-A_1) \rightarrow A_1 = 0.9$$

and then

$$A_{DC} = \frac{B_0 + B_1}{1-A_1} = 1 = \frac{B_0 + B_1}{1-0.9} \rightarrow B_0 + B_1 = 0.1$$

Let's put the zero at infinity so it doesn't affect the transfer function

$$f_{3dB,zero} = \infty = \frac{f_s}{2\pi}\left(1 + \frac{B_1}{B_0}\right) \rightarrow B_0 = 0 \text{ and } B_1 = 0.1$$

A sketch of the filter is seen in Fig. 4.48. ■

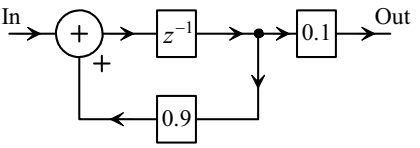


Figure 4.48 First-order digital filter in Ex. 4.11.

General Canonic Form of a Recursive Filter

Before leaving this section, let's show in Fig. 4.49 the general form of an n^{th}-order canonic digital filter (where n indicates the number of poles in the transfer function). The z-domain transfer function of the filter is given by

$$H(z) = \frac{\sum_{i=0}^{m} B_i z^{-i}}{1 - \sum_{i=1}^{n} A_i z^{-i}} = \frac{\sum_{i=0}^{m} B_i z^{n-i}}{z^n - \sum_{i=1}^{n} A_i z^{n-i}} \tag{4.54}$$

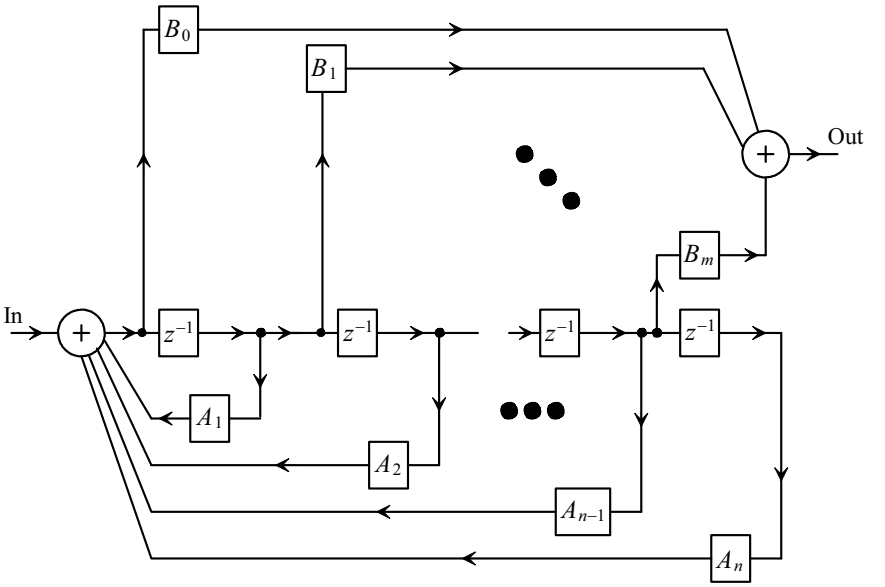
If we want to write the frequency domain transfer function, we write, again assuming $f << f_s$ and using Eq. (4.45),

$$H(f) \approx \frac{\sum_{i=0}^{n-1} B_i \left(1 - j\frac{2\pi f}{f_s}\right)^i}{1 - \sum_{i=1}^{n} A_i \left(1 - j\frac{2\pi f}{f_s}\right)^i} \tag{4.55}$$

or

$$H(f) \approx \frac{\sum_{i=0}^{n-1} B_i \left(1 + j\frac{2\pi f}{f_s}\right)^{n-i}}{\left(1 + j\frac{2\pi f}{f_s}\right)^n - \sum_{i=1}^{n} A_i \left(1 + j\frac{2\pi f}{f_s}\right)^{n-i}} \tag{4.56}$$

While we can design higher-order digital filters using the topology of Fig. 4.49, we will restrict our analysis to first- and second-order filters where hand calculations are relatively easy to perform. Note that we can increase the attenuation of a filter by using several of these sections in cascade.



Number of poles $n \geq$ number of zeroes.

Figure 4.49 General canonic form of a digital filter.

4.3.4 The Biquadratic Transfer Function

The digital biquad filter based on the canonic form seen in Fig. 4.49 is shown in Fig. 4.50. The transfer function of this filter is

$$H(z) = \frac{Y(z)}{X(z)} = \frac{B_0 + B_1 z^{-1} + B_2 z^{-2}}{1 - A_1 z^{-1} - A_2 z^{-2}} = \frac{B_0 z^2 + B_1 z + B_2}{z^2 - A_1 z - A_2} \tag{4.57}$$

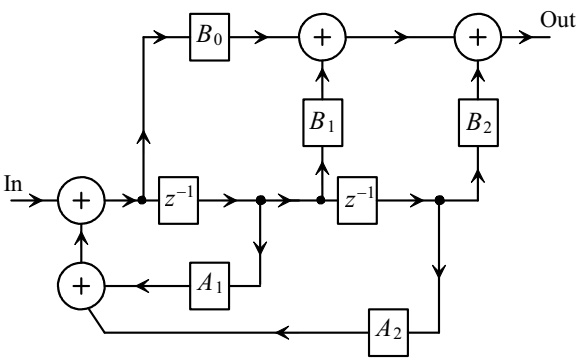


Figure 4.50 The digital biquad filter (see Fig. 4.49).

In order to translate this transfer function into the frequency domain, we use Eq. (4.45) and assume our frequencies of interest are much less than the sampling frequency

$$H(f) = \frac{B_0\left(1+\frac{s}{f_s}\right)^2 + B_1\left(1+\frac{s}{f_s}\right) + B_2}{\left(1+\frac{s}{f_s}\right)^2 - A_1\left(1+\frac{s}{f_s}\right) - A_2} \tag{4.58}$$

After some algebraic manipulation we can put this equation in the form seen in Eq. (3.62)

$$H(f) = \frac{B_0 \cdot s^2 + f_s(2B_0 + B_1) \cdot s + f_s^2(B_0 + B_1 + B_2)}{s^2 + f_s(2 - A_1) \cdot s + f_s^2(1 - A_1 - A_2)} \tag{4.59}$$

where

$$a_2 = B_0 \tag{4.60}$$

$$a_1 = f_s(2B_0 + B_1) \tag{4.61}$$

$$a_0 = f_s^2(B_0 + B_1 + B_2) \tag{4.62}$$

$$\frac{2\pi f_0}{Q} = f_s(2 - A_1) \tag{4.63}$$

and finally,

$$f_0 = \frac{f_s}{2\pi} \cdot \sqrt{(1 - A_1 - A_2)} \tag{4.64}$$

Example 4.12
Repeat Ex. 3.8 using the digital biquad clocked at 100 MHz.

In this example a lowpass filter is designed with f_0 = 1.59 MHz and Q = 0.707. Reviewing Fig. 3.35, we see that for a lowpass filter a_2 and a_1 are zero. This means, in Eq. (4.59), B_0 and B_1 are zero. Further,

$$Q = \frac{2\pi \cdot 1.59 \times 10^6}{100 \times 10^6 (2 - A_1)} = 0.707 \rightarrow A_1 = 1.859$$

and

$$1 - A_1 - A_2 = \left(\frac{2\pi f_0}{f_s}\right)^2 \rightarrow A_2 = -0.869$$

and finally, because the gain at DC is 1,

$$B_2 = 1 - A_1 - A_2 = 0.01$$

Note that if a scaling in the amplitude is allowable, we can remove this multiplication or approximate it with shifts in the digital word.

The simulation results are seen in Fig. 4.51. In order to implement this simulation in SPICE, we used transmission lines for the delay elements and voltage- controlled voltage sources for both the multiplications and the adders. This method allows us to simulate the filter's frequency response. Note that the frequency response is periodic with the filter's clocking frequency. ■

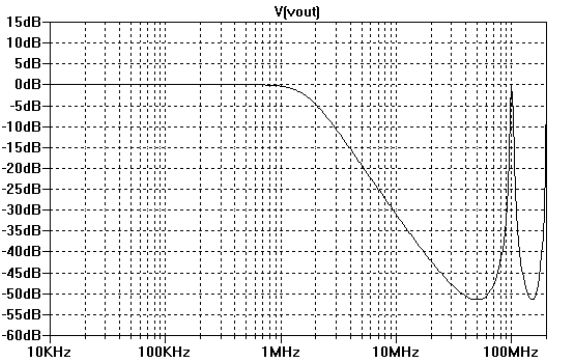


Figure 4.51 Simulating the digital filter in Ex. 4.12

Comparing Biquads to Sinc-Shaped Filters

Consider the frequency response of the Sinc-shaped lowpass filter shown in Fig. 4.52. This filter uses a clocking frequency of 100 MHz and a K of 16 or

$$|H(z)| = \left| \frac{1 - z^{-16}}{1 - z^{-1}} \right|^3 \tag{4.65}$$

Note the significant droop in the filter's response. It's desirable to design a filter that doesn't have this droop or, even more desirable, contains a small amount of peaking to compensate for a Sinc-shaped attenuation from a S/H, decimation filter, etc. Using Eqs. (4.59) - (4.64) we can set, for a digital biquad equivalent of this filter, $B_2 = 0.03125$, $A_1 = 1.75$, $A_2 = -0.78125$ (there are several other solutions, depending on the desired complexity or performance of the filter). The simulation results comparing the Sinc and biquad filters are seen in Fig. 4.53. The transfer function of the biquad filter is

$$H(z) = \frac{0.03125}{z^2 - 1.75z + 0.78125} \tag{4.66}$$

The block diagram of the filter is seen in Fig. 4.54.

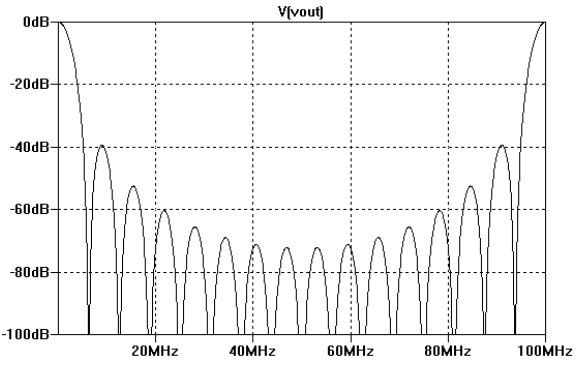


Figure 4.52 Frequency response of a third-order Sinc filter with K = 16.

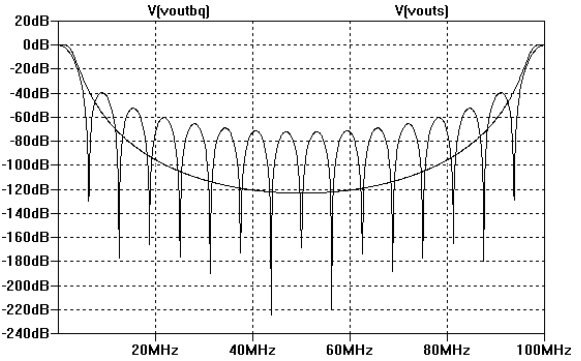


Figure 4.53 Comparing a third-order lowpass Sinc filter to a third-order biquad.

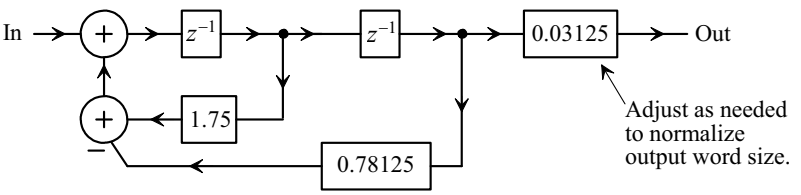


Figure 4.54 The digital biquad filter described by Eq. (4.66).

Example 4.13

Redesign the filter in Fig. 4.54 so that the response has a small amount of peaking at 3.125 MHz. Compare, with simulations, the biquad's response to the Sinc filter's response seen in Fig. 4.52.

Reviewing Eq. (4.63), we see that to increase the Q we need to increase A_1. Keeping in mind that we want to have simple multiplications relying heavily on shifts, let's try increasing A_1 to 1.78125 and A_2 to 0.8125. The simulation results are seen in Fig. 4.55. In this figure we compare the modified filter response to the response of the Sinc filter. Note that we have a couple of dB peaking in the cascaded biquad filter's output. ■

A Comment Concerning Multiplications

While discussing the implementation of digital multipliers is outside the scope of this book a comment is in order about simple multiplier implementations. While, in some filtering applications, simple shifts can prove very useful, we can implement more useful multipliers using adders. Figure 4.56 shows one possible implementation using a single adder along with the associated multiplication factors. We could implement the coefficients in Ex. 4.13 using a similar scheme. For example, $A_1 = 1.78125 = 2 - 0.25 + 0.03125$ and $A_2 = 0.8125 = 1 - 0.25 + 0.0625$ (both requiring two adders). Other creative ways can be used to implement multipliers. For example, a multiplication of 0.5625 by cascading two simple multipliers with multiplication factors 0.75.

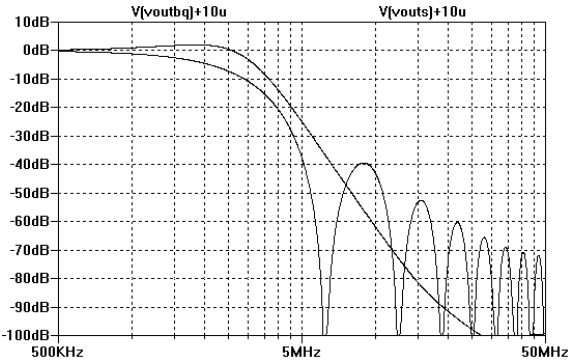
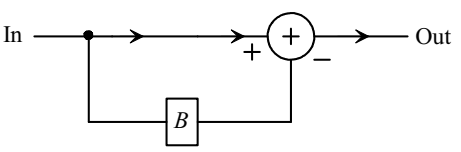


Figure 4.55 Designing a biquad filter with peaking, see Ex. 4.13.



B =	Multiply by
1	0
0.5	0.5
0.25	0.75
0.125	0.875
0.0625	0.9375
0.03125	0.96875

Figure 4.56 A simple multiplier using a single adder.

ADDITIONAL READING

[1] M. Weeks, *Digital Signal Processing Using MATLAB and Wavelets*, Infinity Science Press, 2007. ISBN 978-0977858200

[2] T. B. Welch, C. H. G. Wright, M. G. Morrow, *Real-Time Digital Signal Processing from Matlab to C with the TMS320C6x DSK*, CRC, 2006. ISBN 978-0849373824

[3] S. Haykin and M. Moher, *An Introduction to Analog and Digital Communications*, Second Edition, John Wiley and Sons, 2006. ISBN 978-0471432227

[4] R. G. Lyons, *Understanding Digital Signal Processing*, Second Edition, Prentice-Hall, 2004. ISBN 978-0131089891

[5] P. A. Lynn and W. Fuerst, *Introductory Digital Signal Processing*, Second Edition, John Wiley and Sons, 1998. ISBN 978-0471976318

[6] L. W. Couch, *Modern Communication Systems: Principles and Applications*, Prentice-Hall, 1995. ISBN 978-0023252860

[7] E. P. Cunningham, *Digital Filtering: An Introduction*, John Wiley and Sons, 1995. ISBN 978-0471124757

QUESTIONS

4.1 If V_{REF+} = 1.0 V and V_{REF-} = 0 regenerate Fig. 4.1 using SPICE. (Design a 3-bit ideal DAC model in SPICE.) The y-axis will be voltages in decimal form.

4.2 If, again, V_{REF+} = 1.0 V and V_{REF-} = 0, sketch Fig. 4.1 for a 1-bit DAC. Note that the digital input code will either be a 0 or a 1 and the analog voltage out of the DAC will be either 0 or 1.0 V. Using Eq. (4.1) what is the voltage value of 1 LSB? How does this compare to the value of 1 LSB we get from the sketch? Is Eq. (4.1) valid for a 1-bit DAC? Why? The 1-bit DAC will be a ubiquitous component in our noise-shaping modulators later in the book (see Fig. 7.15).

4.3 Why do the transfer curves of Fig. 4.3 show a shift of 1/2 LSB to the left? How do we implement this shift in SPICE?

4.4 Use SPICE to implement 4-bit ADC and DAC. If the converters are clocked at 100 MHz (and the outputs of the ADC are connected to the inputs of the DAC), apply an input sinewave (to the ADC) that has an amplitude of 500 mV peak centered around 500 mV DC with a frequency of 5 MHz. Again, use V_{REF+} = 1.0 V and V_{REF-} = 0. Show the DAC's analog output.

4.5 Using SPICE generate the spectrums of the input and output signals in question 4.4.

4.6 Suppose we think of the 1-bit input, 0 or 1, in Fig. 4.9 as +1 or −1 (two's complement numbers). What is the output of the digital filter when the input is always 0? Is the magnitude response seen in Fig. 4.10 correct? Why?

4.7 Suppose the 1-bit input signal seen in Fig. 4.9 is an alternating sequence of 101010... In terms of two's complement numbers, what is the output of the digital filter (what is the output of the counter)? What is the frequency of the input signal? Is the frequency response seen in Fig. 4.10 correct?

4.8 Repeat Ex. 4.3 for a filter with a transfer function of

$$\frac{1-z^{-7}}{1-z^{-1}}$$

Also, plot the location of the filter's poles and zeroes in the z-plane.

4.9 Repeat Ex. 4.3 for a filter with a transfer function of

$$\frac{1-z^{-9}}{1-z^{-1}}$$

4.10 Repeat Ex. 4.5 if L is increased to 3.

4.11 Sketch the impulse response of the filter seen in Fig. 4.19.

4.12 What are the transfer functions of the bandpass filters, indicated in Fig. 4.22, with center frequencies of $f_s/6$ and $f_s/3$? Sketch the frequency responses and the location of their poles and zeroes in the z-plane.

4.13 Simulate, using an ideal 8-bit ADC on the input, and an ideal DAC on the output (calculate the size of the DAC), the operation of the digital resonator seen in Fig. 4.23.

4.14 Qualitatively explain why the desired spectrum of an input signal can't be increased by passing data through an interpolator. Using the simulations given in Ex. 4.8, verify that this is indeed the case.

4.15 In Fig. 4.32, which blocks serve as the AAF and which serve as the S/H?

4.16 For the FIR filter seen in Fig. 4.35 with all coefficients set to 0.25, sketch the filter's frequency response.

4.17 For the filter seen in Fig. 4.57 determine the range of values for a and b where the filter will be stable. What is the filter's transfer function? Sketch the location of the filter's poles and zeroes.

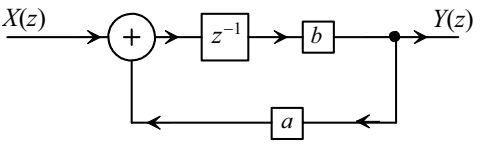


Figure 4.57 A weighted integrating filter.

4.18 Repeat Ex. 4.9 for a filter with a transfer function of

$$\frac{v_{out}}{v_{in}} = \frac{1 + j \cdot \frac{f}{4\ MHz}}{1 + j \cdot \frac{f}{800\ kHz}}$$

4.19 Repeat Question 4.18 using the canonic form of the first-order digital filter.

4.20 Repeat Ex. 4.12 if the Q is increased to 1.

4.21 Show that the filter shown in Fig. 4.58 can be implemented using a single multiplier.

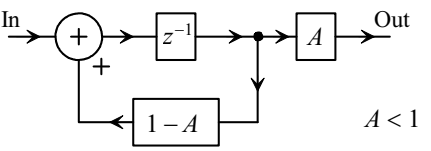


Figure 4.58 Filter used for question 4.21.

4.22 Show that if the values of A and B are restricted to 1, 0.5, 0.25, 0.125, etc. that the circuit of Fig. 4.59 can be used to implement multiplication by coefficients that aren't directly powers of two. How would a multiply by 0.75 be implemented? a multiply-by-0.9375? a multiply-by-0.5625?

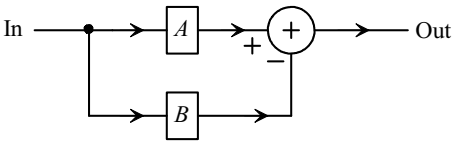


Figure 4.59 A simple multiplier where A and B simply shift the data.