CHAPTER

3

ADC and DAC

Most of the signals directly encountered in science and engineering are *continuous*: light intensity that changes with distance; voltage that varies over time; a chemical reaction rate that depends on temperature, etc. Analog-to-Digital Conversion (ADC) and Digital-to-Analog Conversion (DAC) are the processes that allow digital computers to interact with these everyday signals. Digital information is different from its continuous counterpart in two important respects: it is *sampled*, and it is *quantized*. Both of these restrict how much information a digital signal can contain. This chapter is about *information management*: understanding what information you need to retain, and what information you can afford to lose. In turn, this dictates the selection of the sampling frequency, number of bits, and type of analog filtering needed for converting between the analog and digital realms.

Quantization

First, a bit of trivia. As you know, it is a *digital* computer, not a *digit* computer. The information processed is called *digital* data, not *digit* data. Why then, is analog-to-digital conversion generally called: *digitize and digitization*, rather than *digitalize and digitalization*? The answer is nothing you would expect. When electronics got around to inventing digital techniques, the preferred names had already been snatched up by the medical community nearly a century before. *Digitalize* and *digitalization* mean to administer the heart stimulant *digitalis*.

Figure 3-1 shows the electronic waveforms of a typical analog-to-digital conversion. Figure (a) is the analog signal to be digitized. As shown by the labels on the graph, this signal is a *voltage* that varies over *time*. To make the numbers easier, we will assume that the voltage can vary from 0 to 4.095 volts, corresponding to the digital numbers between 0 and 4095 that will be produced by a 12 bit digitizer. Notice that the block diagram is broken into two sections, the sample-and-hold (S/H), and the analog-to-digital converter (ADC). As you probably learned in electronics classes, the sample-and-hold is required to keep the voltage entering the ADC constant while the

conversion is taking place. However, this is *not* the reason it is shown here; breaking the digitization into these two stages is an important theoretical model for understanding digitization. The fact that it happens to look like common electronics is just a fortunate bonus.

As shown by the difference between (a) and (b), the output of the sample-and-hold is allowed to change only at periodic intervals, at which time it is made identical to the instantaneous value of the input signal. Changes in the input signal that occur between these sampling times are completely ignored. That is, **sampling** converts the *independent variable* (time in this example) from continuous to discrete.

As shown by the difference between (b) and (c), the ADC produces an integer value between 0 and 4095 for each of the flat regions in (b). This introduces an error, since each plateau can be *any* voltage between 0 and 4.095 volts. For example, both 2.56000 volts and 2.56001 volts will be converted into digital number 2560. In other words, **quantization** converts the *dependent variable* (voltage in this example) from continuous to discrete.

Notice that we carefully avoid comparing (a) and (c), as this would lump the sampling and quantization together. It is important that we analyze them separately because they degrade the signal in different ways, as well as being controlled by different parameters in the electronics. There are also cases where one is used without the other. For instance, sampling without quantization is used in switched capacitor filters.

First we will look at the effects of quantization. Any one sample in the digitized signal can have a maximum error of ±½ **LSB** (**Least Significant Bit**, jargon for the distance between adjacent quantization levels). Figure (d) shows the quantization error for this particular example, found by subtracting (b) from (c), with the appropriate conversions. In other words, the digital output (c), is equivalent to the continuous input (b), *plus* a quantization error (d). An important feature of this analysis is that the quantization error appears very much like *random noise*.

This sets the stage for an important model of quantization error. In most cases, *quantization results in nothing more than the addition of a specific amount of random noise to the signal*. The additive noise is uniformly distributed between ±½ LSB, has a mean of zero, and a standard deviation of $1/\sqrt{12}$ LSB (~0.29 LSB). For example, passing an analog signal through an 8 bit digitizer adds an rms noise of: 0.29/256, or about 1/900 of the full scale value. A 12 bit conversion adds a noise of: $0.29/4096 \approx 1/14{,}000$, while a 16 bit conversion adds: $0.29/65536 \approx 1/227{,}000$. Since quantization error is a random noise, the *number of bits* determines the *precision* of the data. For example, you might make the statement: "We increased the precision of the measurement from 8 to 12 bits."

This model is extremely powerful, because the random noise generated by quantization will simply add to whatever noise is already present in the

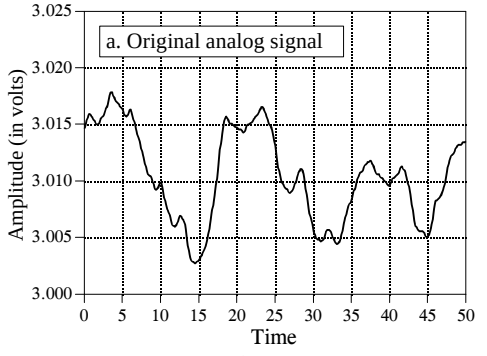
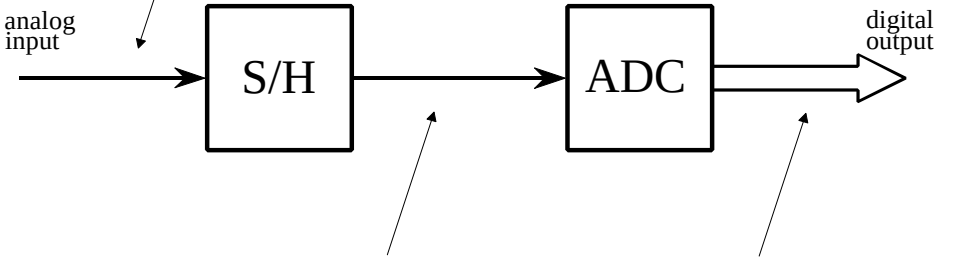
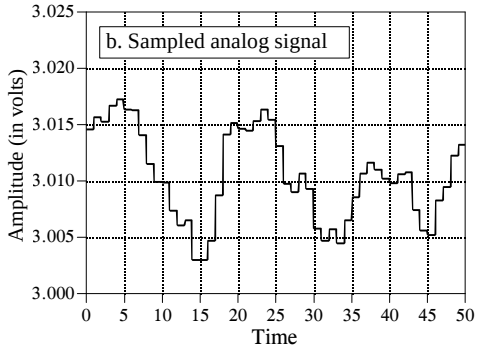
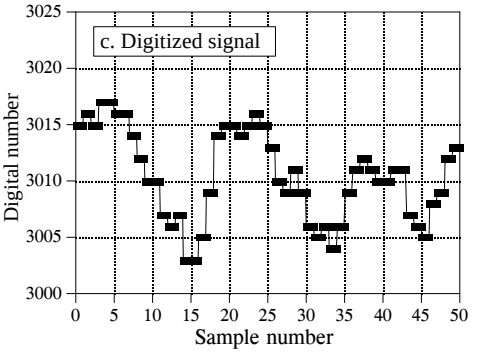
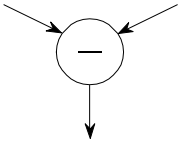


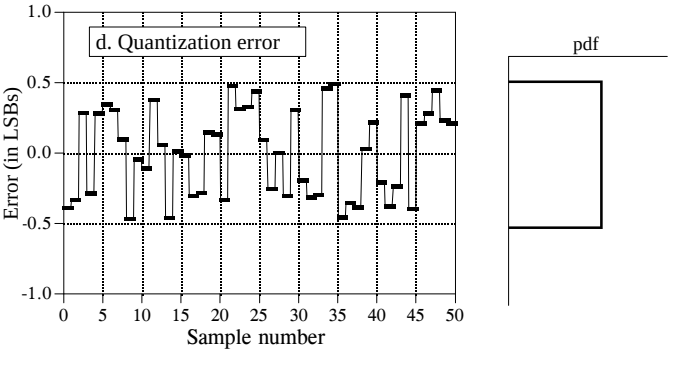
FIGURE 3-1
Waveforms illustrating the digitization process. The conversion is broken into two stages to allow the effects of *sampling* to be separated from the effects of *quantization*. The first stage is the sample-and-hold (S/H), where the only information retained is the instantaneous value of the signal when the periodic sampling takes place. In the second stage, the ADC converts the voltage to the nearest integer number. This results in each sample in the digitized signal having an error of up to ±½ LSB, as shown in (d). As a result, quantization can usually be modeled as simply adding noise to the signal.











analog signal. For example, imagine an analog signal with a maximum amplitude of 1.0 volt, and a random noise of 1.0 millivolt rms. Digitizing this signal to 8 bits results in 1.0 volt becoming digital number 255, and 1.0 millivolt becoming 0.255 LSB. As discussed in the last chapter, random noise signals are combined by adding their *variances*. That is, the signals are added in quadrature: $\sqrt{A^2 + B^2} = C$. The total noise on the digitized signal is therefore given by: $\sqrt{0.255^2 + 0.29^2} = 0.386$ LSB. This is an increase of about 50% over the noise already in the analog signal. Digitizing this same signal to 12 bits would produce virtually no increase in the noise, and *nothing* would be lost due to quantization. When faced with the decision of how many bits are needed in a system, ask two questions: (1) How much noise is *already* present in the analog signal? (2) How much noise can be *tolerated* in the digital signal?

When isn't this model of quantization valid? Only when the quantization error cannot be treated as random. The only common occurrence of this is when the analog signal remains at about the same value for many consecutive samples, as is illustrated in Fig. 3-2a. The output remains *stuck* on the same digital number for many samples in a row, even though the analog signal may be changing up to ±½ LSB. Instead of being an additive random noise, the quantization error now looks like a thresholding effect or weird distortion.

Dithering is a common technique for improving the digitization of these slowly varying signals. As shown in Fig. 3-2b, a small amount of random noise is added to the analog signal. In this example, the added noise is normally distributed with a standard deviation of 2/3 LSB, resulting in a peak-to-peak amplitude of about 3 LSB. Figure (c) shows how the addition of this dithering noise has affected the digitized signal. Even when the original analog signal is changing by less than ±½ LSB, the added noise causes the digital output to randomly toggle between adjacent levels.

To understand how this improves the situation, imagine that the input signal is a constant analog voltage of 3.0001 volts, making it one-tenth of the way between the digital levels 3000 and 3001. Without dithering, taking 10,000 samples of this signal would produce 10,000 identical numbers, all having the value of 3000. Next, repeat the thought experiment with a small amount of dithering noise added. The 10,000 values will now oscillate between two (or more) levels, with about 90% having a value of 3000, and 10% having a value of 3001. Taking the average of all 10,000 values results in something close to 3000.1. Even though a single measurement has the inherent ±½ LSB limitation, the statistics of a large number of the samples can do much better. This is quite a strange situation: *adding noise provides more information*.

Circuits for dithering can be quite sophisticated, such as using a computer to generate random numbers, and then passing them through a DAC to produce the added noise. After digitization, the computer can *subtract*

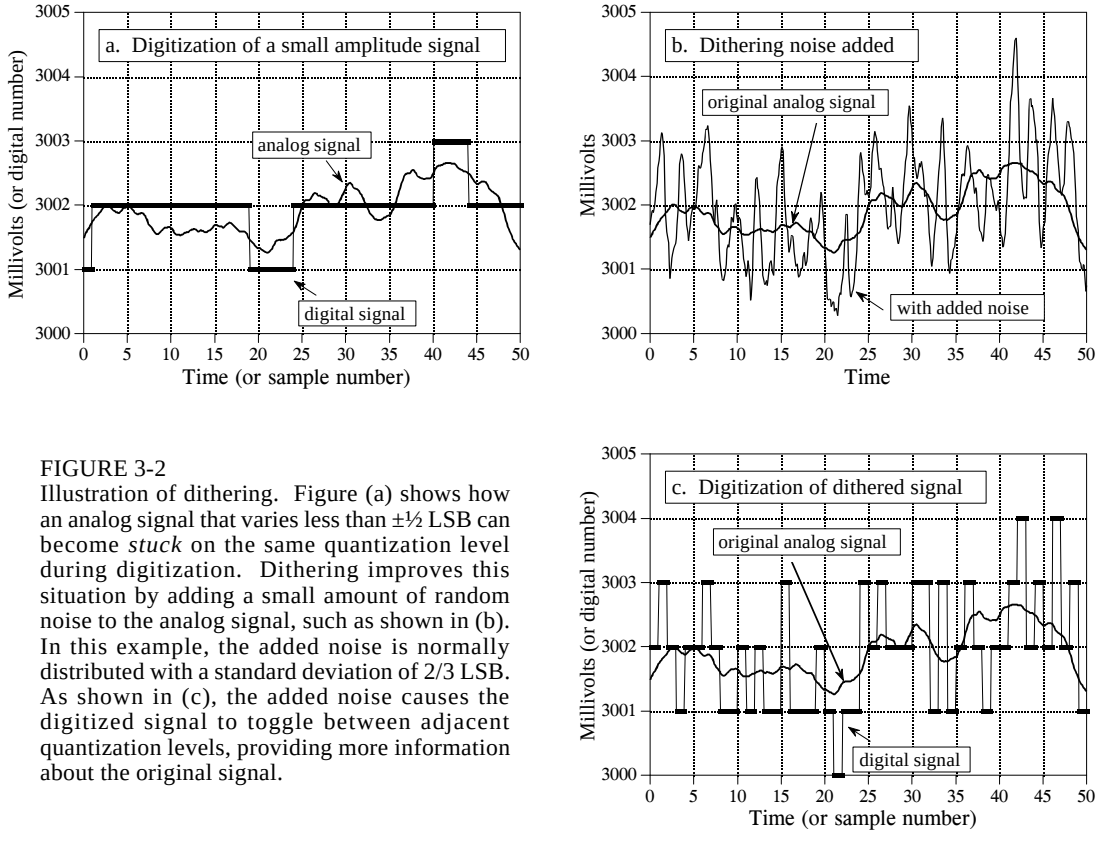


FIGURE 3-2
Illustration of dithering. Figure (a) shows how an analog signal that varies less than ±½ LSB can become *stuck* on the same quantization level during digitization. Dithering improves this situation by adding a small amount of random noise to the analog signal, such as shown in (b). In this example, the added noise is normally distributed with a standard deviation of 2/3 LSB. As shown in (c), the added noise causes the digitized signal to toggle between adjacent quantization levels, providing more information about the original signal.

the random numbers from the digital signal using floating point arithmetic. This elegant technique is called **subtractive dither**, but is only used in the most elaborate systems. The simplest method, although not always possible, is to use the noise already present in the analog signal for dithering.

The Sampling Theorem

The definition of *proper sampling* is quite simple. Suppose you sample a continuous signal in some manner. If you can exactly *reconstruct* the analog signal from the samples, you must have done the sampling *properly*. Even if the sampled data appears confusing or incomplete, the key information has been captured if you can reverse the process.

Figure 3-3 shows several sinusoids before and after digitization. The continuous line represents the analog signal entering the ADC, while the square markers are the digital signal leaving the ADC. In (a), the analog signal is a constant DC value, a cosine wave of *zero* frequency. Since the analog signal is a series of straight lines between each of the samples, all of the information needed to reconstruct the analog signal is contained in the digital data. According to our definition, this is *proper sampling*.

The sine wave shown in (b) has a frequency of 0.09 of the sampling rate. This might represent, for example, a 90 cycle/second sine wave being sampled at 1000 samples/second. Expressed in another way, there are 11.1 samples taken over each complete cycle of the sinusoid. This situation is more complicated than the previous case, because the analog signal cannot be reconstructed by simply drawing straight lines between the data points. Do these samples properly represent the analog signal? The answer is yes, because no other sinusoid, or combination of sinusoids, will produce this pattern of samples (within the reasonable constraints listed below). These samples correspond to only one analog signal, and therefore the analog signal can be exactly reconstructed. Again, an instance of *proper sampling*.

In (c), the situation is made more difficult by increasing the sine wave's frequency to 0.31 of the sampling rate. This results in only 3.2 samples per sine wave cycle. Here the samples are so sparse that they don't even appear to follow the general trend of the analog signal. Do these samples properly represent the analog waveform? Again, the answer is yes, and for exactly the same reason. The samples are a unique representation of the analog signal. All of the information needed to reconstruct the continuous waveform is contained in the digital data. How you go about doing this will be discussed later in this chapter. Obviously, it must be more sophisticated than just drawing straight lines between the data points. As strange as it seems, this is *proper sampling* according to our definition.

In (d), the analog frequency is pushed even higher to 0.95 of the sampling rate, with a mere 1.05 samples per sine wave cycle. Do these samples properly represent the data? *No, they don't!* The samples represent a *different* sine wave from the one contained in the analog signal. In particular, the original sine wave of 0.95 frequency misrepresents itself as a sine wave of 0.05 frequency in the digital signal. This phenomenon of sinusoids changing frequency during sampling is called **aliasing**. Just as a criminal might take on an assumed name or identity (an *alias*), the sinusoid assumes another frequency that is not its own. Since the digital data is no longer uniquely related to a particular analog signal, an unambiguous reconstruction is impossible. There is nothing in the sampled data to suggest that the original analog signal had a frequency of 0.95 rather than 0.05. The sine wave has hidden its true identity completely; the perfect crime has been committed! According to our definition, this is an example of *improper sampling*.

This line of reasoning leads to a milestone in DSP, the **sampling theorem**. Frequently this is called the *Shannon* sampling theorem, or the *Nyquist* sampling theorem, after the authors of 1940s papers on the topic. The sampling theorem indicates that a continuous signal can be properly sampled, *only if it does not contain frequency components above one-half of the sampling rate*. For instance, a sampling rate of 2,000 samples/second requires the analog signal to be composed of frequencies below 1000 cycles/second. If frequencies above this limit *are* present in the signal, they will be aliased to frequencies between 0 and 1000 cycles/second, combining with whatever information that was legitimately there.

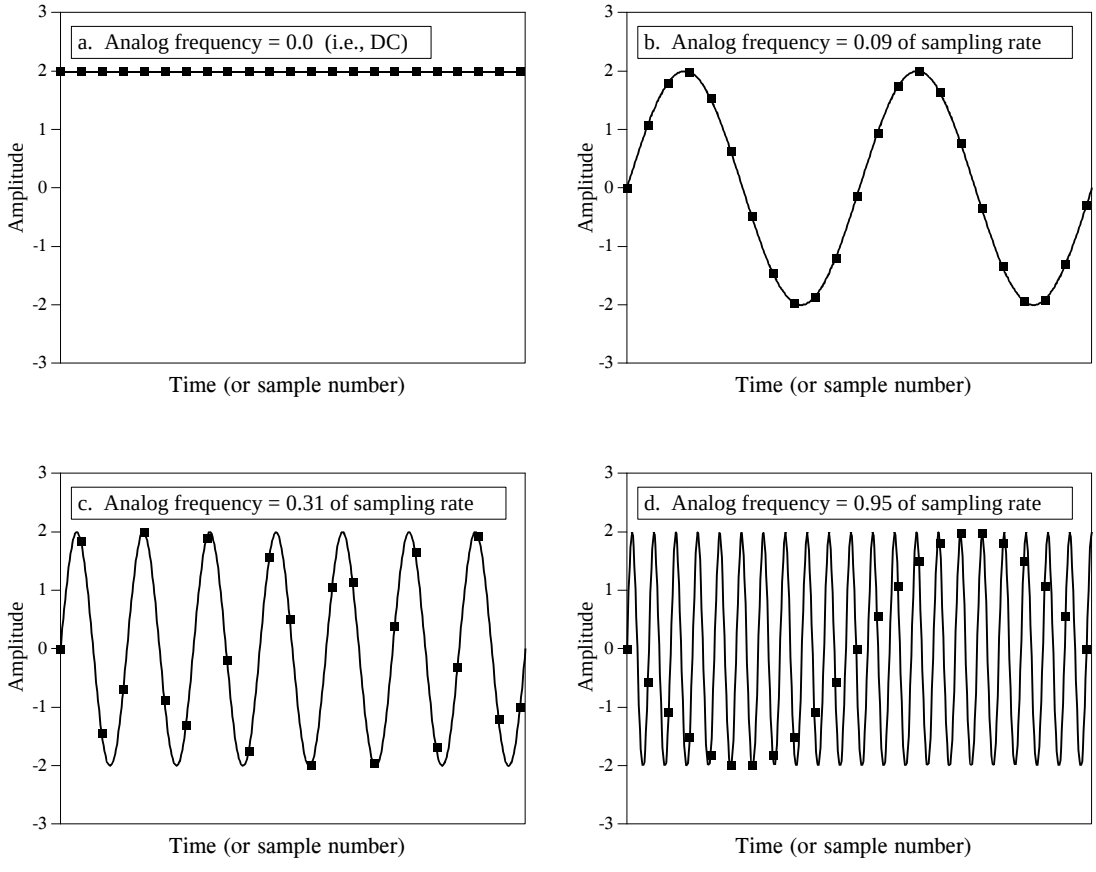


FIGURE 3-3
Illustration of proper and improper sampling. A continuous signal is sampled *properly* if the samples contain all the information needed to recreate the original waveform. Figures (a), (b), and (c) illustrate *proper sampling* of three sinusoidal waves. This is certainly not obvious, since the samples in (c) do not even appear to capture the shape of the waveform. Nevertheless, each of these continuous signals forms a unique one-to-one pair with its pattern of samples. This guarantees that reconstruction can take place. In (d), the frequency of the analog sine wave is greater than the Nyquist frequency (one-half of the sampling rate). This results in *aliasing*, where the frequency of the sampled data is different from the frequency of the continuous signal. Since aliasing has corrupted the information, the original signal cannot be reconstructed from the samples.

Two terms are widely used when discussing the sampling theorem: the **Nyquist frequency** and the **Nyquist rate**. Unfortunately, their meaning is not standardized. To understand this, consider an analog signal composed of frequencies between DC and 3 kHz. To properly digitize this signal it must be sampled at 6,000 samples/sec (6 kHz) or higher. Suppose we choose to sample at 8,000 samples/sec (8 kHz), allowing frequencies between DC and 4 kHz to be properly represented. In this situation there are four important frequencies: (1) the highest frequency in the signal, 3 kHz; (2) twice this frequency, 6 kHz; (3) the sampling rate, 8 kHz; and (4) one-half the sampling rate, 4 kHz. Which of these four is the *Nyquist frequency* and which is the *Nyquist rate*? It depends who you ask! All of the possible combinations are

used. Fortunately, most authors are careful to define how they are using the terms. In this book, they are both used to mean *one-half the sampling rate*.

Figure 3-4 shows how frequencies are changed during aliasing. The key point to remember is that a digital signal *cannot* contain frequencies above one-half the sampling rate (i.e., the Nyquist frequency/rate). When the frequency of the continuous wave is below the Nyquist rate, the frequency of the sampled data is a match. However, when the continuous signal's frequency is above the Nyquist rate, aliasing *changes* the frequency into something that *can* be represented in the sampled data. As shown by the zigzagging line in Fig. 3-4, every continuous frequency above the Nyquist rate has a corresponding digital frequency between zero and one-half the sampling rate. If there happens to be a sinusoid already at this lower frequency, the aliased signal will add to it, resulting in a loss of information. Aliasing is a double curse; information can be lost about the higher *and* the lower frequency. Suppose you are given a digital signal containing a frequency of 0.2 of the sampling rate. If this signal were obtained by proper sampling, the original analog signal *must* have had a frequency of 0.2. If aliasing took place during sampling, the digital frequency of 0.2 could have come from any one of an infinite number of frequencies in the analog signal: 0.2, 0.8, 1.2, 1.8, 2.2, ··· .

Just as aliasing can change the frequency during sampling, it can also change the *phase*. For example, look back at the aliased signal in Fig. 3-3d. The aliased digital signal is *inverted* from the original analog signal; one is a sine wave while the other is a negative sine wave. In other words, aliasing has changed the frequency *and* introduced a 180° phase shift. Only two phase shifts are possible: 0° (no phase shift) and 180° (inversion). The zero phase shift occurs for analog frequencies of 0 to 0.5, 1.0 to 1.5, 2.0 to 2.5, etc. An inverted phase occurs for analog frequencies of 0.5 to 1.0, 1.5 to 2.0, 2.5 to 3.0, and so on.

Now we will dive into a more detailed analysis of sampling and how aliasing occurs. Our overall goal is to understand what happens to the information when a signal is converted from a continuous to a discrete form. The problem is, these are very different things; one is a *continuous waveform* while the other is an *array of numbers*. This "apples-to-oranges" comparison makes the analysis very difficult. The solution is to introduce a theoretical concept called the **impulse train**.

Figure 3-5a shows an example analog signal. Figure (c) shows the signal sampled by using an *impulse train*. The impulse train is a continuous signal consisting of a series of narrow spikes (impulses) that match the original signal at the sampling instants. Each impulse is infinitesimally narrow, a concept that will be discussed in Chapter 13. Between these sampling times the value of the waveform is zero. Keep in mind that the impulse train is a *theoretical* concept, not a waveform that can exist in an electronic circuit. Since both the original analog signal and the impulse train are continuous waveforms, we can make an "apples-apples" comparison between the two.

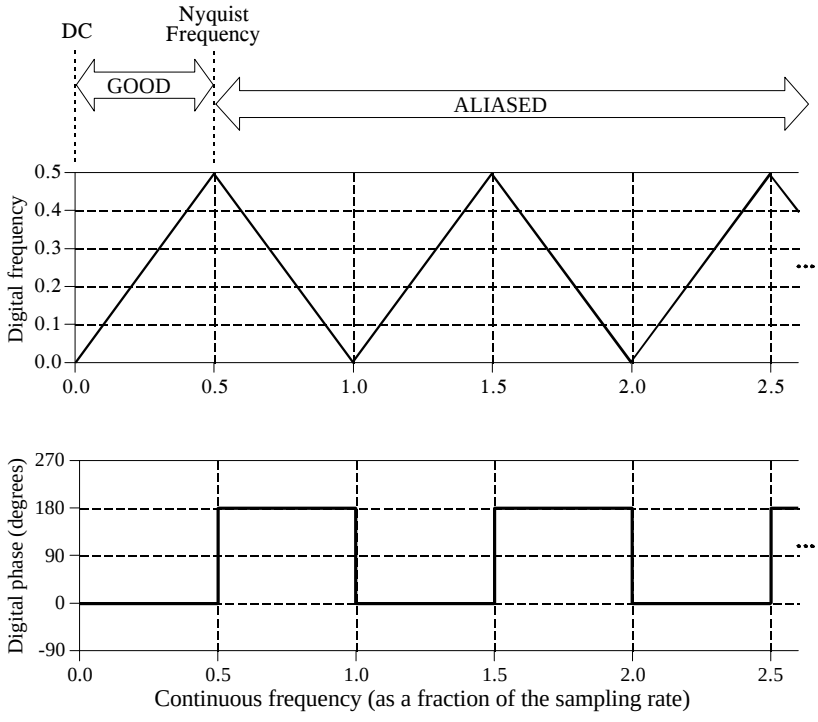


FIGURE 3-4
Conversion of analog frequency into digital frequency during sampling. Continuous signals with a frequency less than one-half of the sampling rate are directly converted into the corresponding digital frequency. Above one-half of the sampling rate, aliasing takes place, resulting in the frequency being misrepresented in the digital data. Aliasing always changes a higher frequency into a lower frequency between 0 and 0.5. In addition, aliasing may also change the phase of the signal by 180 degrees.

Now we need to examine the relationship between the impulse train and the discrete signal (an array of numbers). This one is easy; in terms of *information content*, they are *identical*. If one is known, it is trivial to calculate the other. Think of these as different ends of a bridge crossing between the analog and digital worlds. This means we have achieved our overall goal once we understand the consequences of changing the waveform in Fig. 3-5a into the waveform in Fig. 3.5c.

Three continuous waveforms are shown in the left-hand column in Fig. 3-5. The corresponding *frequency spectra* of these signals are displayed in the right-hand column. This should be a familiar concept from your knowledge of electronics; every waveform can be viewed as being composed of sinusoids of varying amplitude and frequency. Later chapters will discuss the frequency domain in detail. (You may want to revisit this discussion after becoming more familiar with frequency spectra).

Figure (a) shows an analog signal we wish to sample. As indicated by its frequency spectrum in (b), it is composed only of frequency components between 0 and about 0.33 f_s, where f_s is the sampling frequency we intend to

use. For example, this might be a speech signal that has been filtered to remove all frequencies above 3.3 kHz. Correspondingly, f_s would be 10 kHz (10,000 samples/second), our intended sampling rate.

Sampling the signal in (a) by using an impulse train produces the signal shown in (c), and its frequency spectrum shown in (d). This spectrum is a *duplication* of the spectrum of the original signal. Each multiple of the sampling frequency, f_s, $2f_s$, $3f_s$, $4f_s$, etc., has received a *copy* and a *left-for-right flipped copy* of the original frequency spectrum. The copy is called the **upper sideband**, while the flipped copy is called the **lower sideband**. Sampling has generated *new* frequencies. Is this proper sampling? The answer is yes, because the signal in (c) can be transformed back into the signal in (a) by eliminating all frequencies above $\frac{1}{2}f_s$. That is, an analog low-pass filter will convert the impulse train, (b), back into the original analog signal, (a).

If you are already familiar with the basics of DSP, here is a more technical explanation of why this spectral duplication occurs. (Ignore this paragraph if you are new to DSP). In the time domain, sampling is achieved by multiplying the original signal by an impulse train of *unity amplitude* spikes. The frequency spectrum of this unity amplitude impulse train is also a unity amplitude impulse train, with the spikes occurring at multiples of the sampling frequency, f_s, $2f_s$, $3f_s$, $4f_s$, etc. When two time domain signals are multiplied, their frequency spectra are convolved. This results in the original spectrum being duplicated to the location of each spike in the impulse train's spectrum. Viewing the original signal as composed of both positive and negative frequencies accounts for the upper and lower sidebands, respectively. This is the same as amplitude modulation, discussed in Chapter 10.

Figure (e) shows an example of *improper sampling*, resulting from too low of sampling rate. The analog signal still contains frequencies up to 3.3 kHz, but the sampling rate has been lowered to 5 kHz. Notice that f_S, $2f_S$, $3f_S$ ··· along the horizontal axis are spaced closer in (f) than in (d). The frequency spectrum, (f), shows the problem: the duplicated portions of the spectrum have invaded the band between zero and one-half of the sampling frequency. Although (f) shows these overlapping frequencies as retaining their separate identity, in actual practice they add together forming a single confused mess. Since there is no way to separate the overlapping frequencies, information is lost, and the original signal cannot be reconstructed. This overlap occurs when the analog signal contains frequencies greater than one-half the sampling rate, that is, we have proven the sampling theorem.

Digital-to-Analog Conversion

In theory, the simplest method for digital-to-analog conversion is to pull the samples from memory and convert them into an *impulse train*. This is

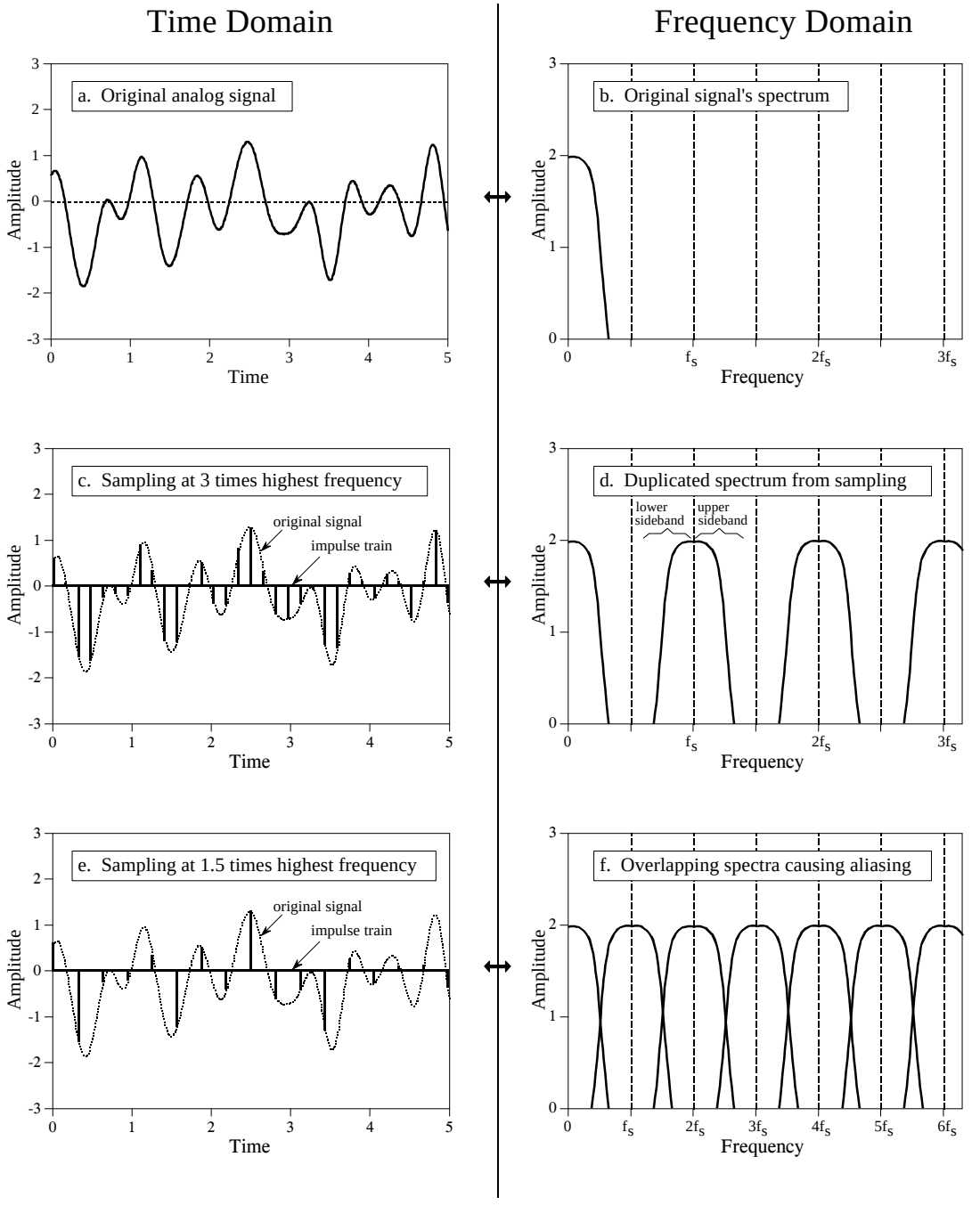


FIGURE 3-5
The sampling theorem in the time and frequency domains. Figures (a) and (b) show an analog signal composed of frequency components between zero and 0.33 of the sampling frequency, f_s. In (c), the analog signal is sampled by converting it to an impulse train. In the frequency domain, (d), this results in the spectrum being duplicated into an infinite number of upper and lower sidebands. Since the original frequencies in (b) exist undistorted in (d), proper sampling has taken place. In comparison, the analog signal in (e) is sampled at 0.66 of the sampling frequency, a value exceeding the Nyquist rate. This results in aliasing, indicated by the sidebands in (f) overlapping.

illustrated in Fig. 3-6a, with the corresponding frequency spectrum in (b). As just described, the original analog signal can be perfectly reconstructed by passing this impulse train through a low-pass filter, with the cutoff frequency equal to one-half of the sampling rate. In other words, the original signal and the impulse train have identical frequency spectra below the Nyquist frequency (one-half the sampling rate). At higher frequencies, the impulse train contains a duplication of this information, while the original analog signal contains nothing (assuming aliasing did not occur).

While this method is mathematically pure, it is difficult to generate the required narrow pulses in electronics. To get around this, nearly all DACs operate by holding the last value until another sample is received. This is called a **zeroth-order hold**, the DAC equivalent of the sample-and-hold used during ADC. (A first-order hold is straight lines between the points, a second-order hold uses parabolas, etc.). The zeroth-order hold produces the staircase appearance shown in (c).

In the frequency domain, the zeroth-order hold results in the spectrum of the impulse train being *multiplied* by the dark curve shown in (d), given by the equation:

EQUATION 3-1
High frequency amplitude reduction due to the zeroth-order hold. This curve is plotted in Fig. 3-6d. The sampling frequency is represented by f_S. For $f = 0$, $H(f) = 1$.

$$H(f) = \left| \frac{\sin(\pi f/f_s)}{\pi f/f_s} \right|$$

This is of the general form: $\sin(\pi x)/(\pi x)$, called the **sinc function** or **sinc(x)**. The sinc function is very common in DSP, and will be discussed in more detail in later chapters. If you already have a background in this material, the zeroth-order hold can be understood as the convolution of the impulse train with a rectangular pulse, having a width equal to the sampling period. This results in the frequency domain being *multiplied* by the Fourier transform of the rectangular pulse, i.e., the sinc function. In Fig. (d), the light line shows the frequency spectrum of the impulse train (the "correct" spectrum), while the dark line shows the sinc. The frequency spectrum of the zeroth order hold signal is equal to the product of these two curves.

The analog filter used to convert the zeroth-order hold signal, (c), into the reconstructed signal, (f), needs to do two things: (1) remove all frequencies above one-half of the sampling rate, and (2) boost the frequencies by the reciprocal of the zeroth-order hold's effect, i.e., *1/sinc(x)*. This amounts to an amplification of about 36% at one-half of the sampling frequency. Figure (e) shows the ideal frequency response of this analog filter.

This *1/sinc(x)* frequency boost can be handled in four ways: (1) ignore it and accept the consequences, (2) design an analog filter to include the *1/sinc(x)*

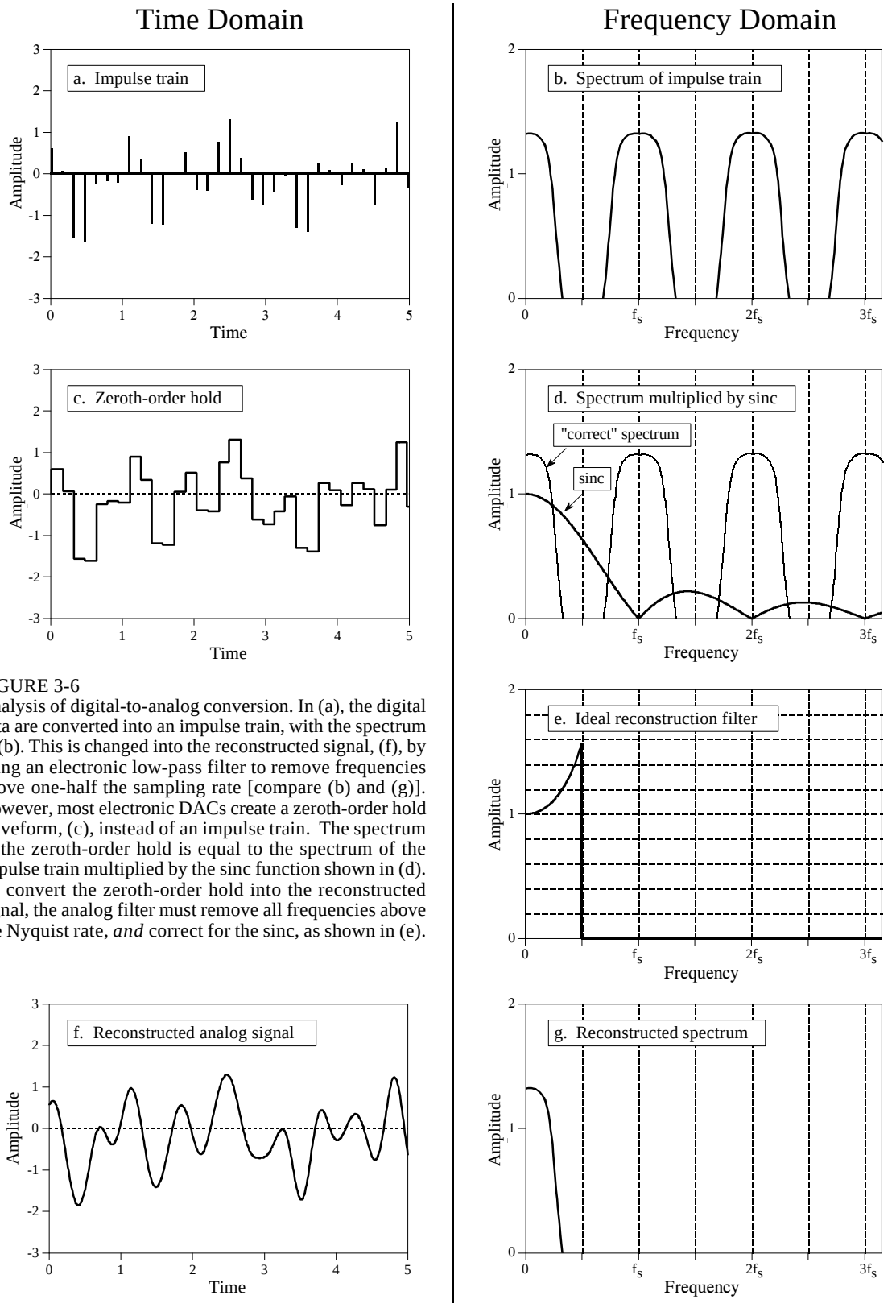


FIGURE 3-6
Analysis of digital-to-analog conversion. In (a), the digital data are converted into an impulse train, with the spectrum in (b). This is changed into the reconstructed signal, (f), by using an electronic low-pass filter to remove frequencies above one-half the sampling rate [compare (b) and (g)]. However, most electronic DACs create a zeroth-order hold waveform, (c), instead of an impulse train. The spectrum of the zeroth-order hold is equal to the spectrum of the impulse train multiplied by the sinc function shown in (d). To convert the zeroth-order hold into the reconstructed signal, the analog filter must remove all frequencies above the Nyquist rate, *and* correct for the sinc, as shown in (e).

response, (3) use a fancy *multirate* technique described later in this chapter, or (4) make the correction in software before the DAC (see Chapter 24).

Before leaving this section on sampling, we need to dispel a common myth about analog versus digital signals. As this chapter has shown, the amount of information carried in a digital signal is limited in two ways: First, the number of bits per sample limits the resolution of the *dependent* variable. That is, small changes in the signal's amplitude may be lost in the quantization noise. Second, the sampling rate limits the resolution of the *independent variable*, i.e., closely spaced events in the analog signal may be lost between the samples. This is another way of saying that frequencies above one-half the sampling rate are lost.

Here is the myth: "Since analog signals use continuous parameters, they have infinitely good resolution in both the independent and the dependent variables." Not true! Analog signals are limited by the same two problems as digital signals: *noise* and *bandwidth* (the highest frequency allowed in the signal). The noise in an analog signal limits the measurement of the waveform's amplitude, just as quantization noise does in a digital signal. Likewise, the ability to separate closely spaced events in an analog signal depends on the highest frequency allowed in the waveform. To understand this, imagine an analog signal containing two closely spaced pulses. If we place the signal through a low-pass filter (removing the high frequencies), the pulses will blur into a single blob. For instance, an analog signal formed from frequencies between DC and 10 kHz will have *exactly* the same resolution as a digital signal sampled at 20 kHz. It must, since the sampling theorem guarantees that the two contain the same information.

Analog Filters for Data Conversion

Figure 3-7 shows a block diagram of a DSP system, *as the sampling theorem dictates it should be*. Before encountering the analog-to-digital converter,

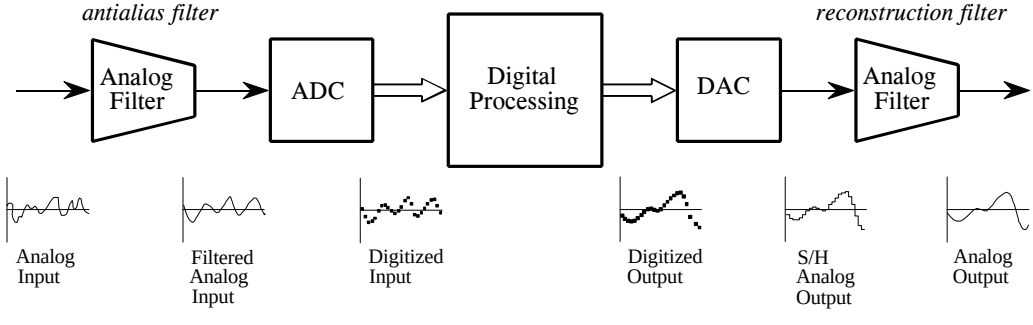


FIGURE 3-7
Analog electronic filters used to comply with the sampling theorem. The electronic filter placed before an ADC is called an *antialias filter*. It is used to remove frequency components above one-half of the sampling rate that would alias during the sampling. The electronic filter placed after a DAC is called a *reconstruction filter*. It also eliminates frequencies above the Nyquist rate, and may include a correction for the zeroth-order hold.

the input signal is processed with an electronic low-pass filter to remove all frequencies above the Nyquist frequency (one-half the sampling rate). This is done to prevent aliasing during sampling, and is correspondingly called an **antialias filter**. On the other end, the digitized signal is passed through a digital-to-analog converter and another low-pass filter set to the Nyquist frequency. This output filter is called a **reconstruction filter**, and may include the previously described zeroth-order-hold frequency boost. Unfortunately, there is a serious problem with this simple model: the limitations of electronic filters can be as bad as the problems they are trying to prevent.

If your main interest is in software, you are probably thinking that you don't need to read this section. *Wrong*! Even if you have vowed never to touch an oscilloscope, an understanding of the properties of analog filters is important for successful DSP. First, the characteristics of every digitized signal you encounter will depend on what type of antialias filter was used when it was acquired. If you don't understand the nature of the antialias filter, you cannot understand the nature of the digital signal. Second, the future of DSP is to replace *hardware* with *software*. For example, the *multirate* techniques presented later in this chapter reduce the need for antialias and reconstruction filters by fancy software tricks. If you don't understand the hardware, you cannot design software to replace it. Third, much of DSP is related to digital filter design. A common strategy is to start with an equivalent *analog filter*, and convert it into software. Later chapters assume you have a basic knowledge of analog filter techniques.

Three types of analog filters are commonly used: **Chebyshev**, **Butterworth**, and **Bessel** (also called a Thompson filter). Each of these is designed to optimize a different performance parameter. The complexity of each filter can be adjusted by selecting the number of **poles** and **zeros**, mathematical terms that will be discussed in later chapters. The more poles in a filter, the more electronics it requires, and the better it performs. Each of these names describe what the filter *does*, not a particular arrangement of resistors and capacitors. For example, a six pole Bessel filter can be implemented by many different types of circuits, all of which have the same overall characteristics. For DSP purposes, the characteristics of these filters are more important than how they are constructed. Nevertheless, we will start with a short segment on the electronic design of these filters to provide an overall framework.

Figure 3-8 shows a common building block for analog filter design, the modified Sallen-Key circuit. This is named after the authors of a 1950s paper describing the technique. The circuit shown is a two pole low-pass filter that can be configured as any of the three basic types. Table 3-1 provides the necessary information to select the appropriate resistors and capacitors. For example, to design a 1 kHz, 2 pole Butterworth filter, Table 3-1 provides the parameters: $k_1 = 0.1592$ and $k_2 = 0.586$. Arbitrarily selecting $R_1 = 10K$ and $C = 0.01uF$ (common values for op amp circuits), R and R_f can be calculated as 15.95K and 5.86K, respectively. Rounding these last two values to the nearest 1% standard resistors, results in $R = 15.8K$ and $R_f = 5.90K$ All of the components should be 1% precision or better.

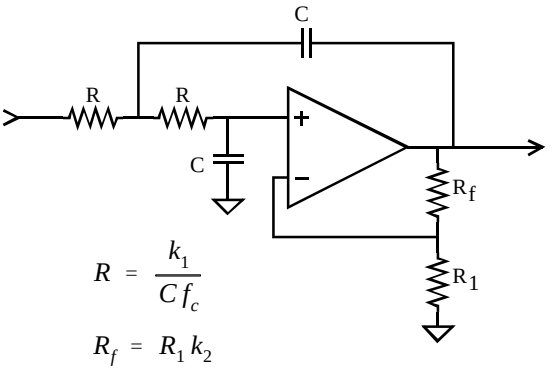


FIGURE 3-8
The modified Sallen-Key circuit, a building block for active filter design. The circuit shown implements a 2 pole low-pass filter. Higher order filters (more poles) can be formed by cascading stages. Find k_1 and k_2 from Table 3-1, arbitrarily select R_1 and C (try 10K and 0.01µF), and then calculate R and R_f from the equations in the figure. The parameter, f_c, is the cutoff frequency of the filter, in hertz.

TABLE 3-1
Parameters for designing Bessel, Butterworth, and Chebyshev (6% ripple) filters.

# poles		Bessel		Butterworth		Chebyshev	
		k_1	k_2	k_1	k_2	k_1	k_2
2	stage 1	0.1251	0.268	0.1592	0.586	0.1293	0.842
4	stage 1	0.1111	0.084	0.1592	0.152	0.2666	0.582
	stage 2	0.0991	0.759	0.1592	1.235	0.1544	1.660
6	stage 1	0.0990	0.040	0.1592	0.068	0.4019	0.537
	stage 2	0.0941	0.364	0.1592	0.586	0.2072	1.448
	stage 3	0.0834	1.023	0.1592	1.483	0.1574	1.846
8	stage 1	0.0894	0.024	0.1592	0.038	0.5359	0.522
	stage 2	0.0867	0.213	0.1592	0.337	0.2657	1.379
	stage 3	0.0814	0.593	0.1592	0.889	0.1848	1.711
	stage 4	0.0726	1.184	0.1592	1.610	0.1582	1.913

The particular op amp used isn't critical, as long as the unity gain frequency is more than 30 to 100 times higher than the filter's cutoff frequency. This is an easy requirement as long as the filter's cutoff frequency is below about 100 kHz.

Four, six, and eight pole filters are formed by cascading 2,3, and 4 of these circuits, respectively. For example, Fig. 3-9 shows the schematic of a 6 pole

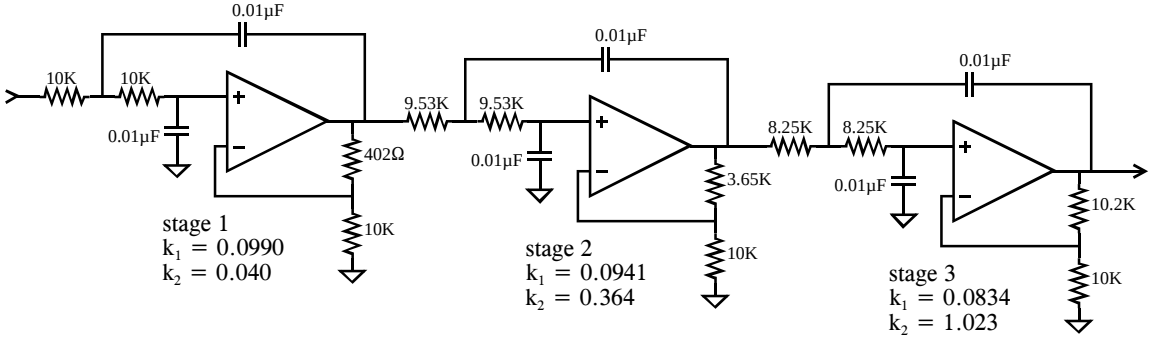


FIGURE 3-9
A six pole Bessel filter formed by cascading three Sallen-Key circuits. This is a low-pass filter with a cutoff frequency of 1 kHz.

Bessel filter created by cascading three stages. Each stage has different values for k_1 and k_2 as provided by Table 3-1, resulting in different resistors and capacitors being used. Need a high-pass filter? Simply swap the R and C components in the circuits (leaving R_f and R_1 alone).

This type of circuit is very common for small quantity manufacturing and R&D applications; however, serious production requires the filter to be made as an *integrated circuit*. The problem is, it is difficult to make resistors directly in silicon. The answer is the **switched capacitor filter**. Figure 3-10 illustrates its operation by comparing it to a simple RC network. If a step function is fed into an RC low-pass filter, the output rises exponentially until it matches the input. The voltage on the capacitor doesn't change instantaneously, because the resistor restricts the flow of electrical charge.

The switched capacitor filter operates by replacing the basic resistor-capacitor network with two capacitors and an electronic switch. The newly added capacitor is much smaller in value than the already existing capacitor, say, 1% of its value. The switch alternately connects the small capacitor between the input and the output at a very high frequency, typically 100 times faster than the cutoff frequency of the filter. When the switch is connected to the input, the small capacitor rapidly charges to whatever voltage is presently on the input. When the switch is connected to the output, the charge on the small capacitor is transferred to the large capacitor. In a resistor, the rate of charge transfer is determined by its resistance. In a switched capacitor circuit, the rate of charge transfer is determined by the value of the small capacitor *and* by the switching frequency. This results in a very useful feature of switched capacitor

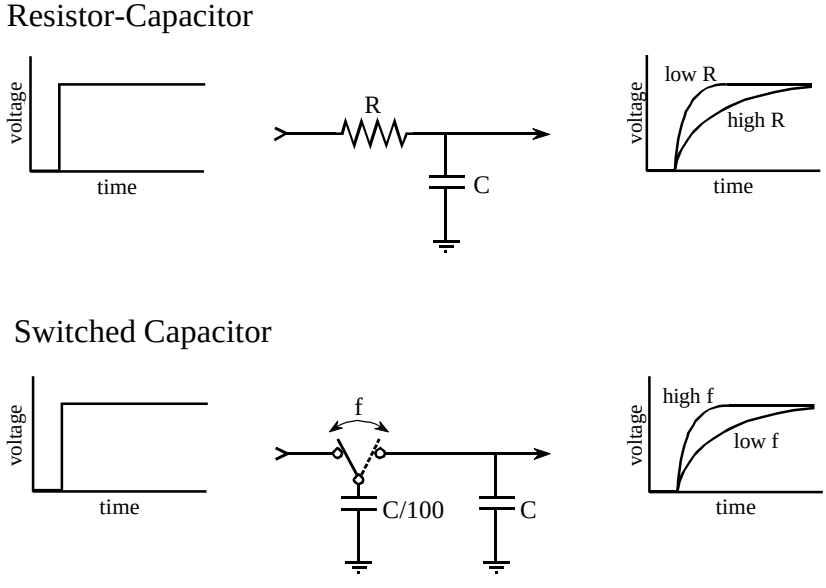


FIGURE 3-10
Switched capacitor filter operation. Switched capacitor filters use switches and capacitors to mimic resistors. As shown by the equivalent step responses, two capacitors and one switch can perform the same function as a resistor-capacitor network.

filters: *the cutoff frequency of the filter is directly proportional to the clock frequency used to drive the switches*. This makes the switched capacitor filter ideal for data acquisition systems that operate with more than one sampling rate. These are easy-to-use devices; pay ten bucks and have the performance of an eight pole filter inside a single 8 pin IC.

Now for the important part: the characteristics of the three classic filter types. The first performance parameter we want to explore is **cutoff frequency sharpness**. A low-pass filter is designed to block all frequencies above the cutoff frequency (the **stopband**), while passing all frequencies below (the **passband**). Figure 3-11 shows the frequency response of these three filters on a logarithmic (dB) scale. These graphs are shown for filters with a one hertz cutoff frequency, but they can be directly scaled to whatever cutoff frequency you need to use. How do these filters rate? The Chebyshev is clearly the best, the Butterworth is worse, and the Bessel is absolutely ghastly! As you probably surmised, this is what the Chebyshev is designed to do, **roll-off** (drop in amplitude) as rapidly as possible.

Unfortunately, even an 8 pole Chebyshev isn't as good as you would like for an antialias filter. For example, imagine a 12 bit system sampling at 10,000 samples per second. The sampling theorem dictates that any frequency above 5 kHz will be aliased, something you want to avoid. With a little guess work, you decide that all frequencies above 5 kHz must be reduced in amplitude by a factor of 100, insuring that any aliased frequencies will have an amplitude of less than one percent. Looking at Fig. 3-11c, you find that an 8 pole Chebyshev filter, with a cutoff frequency of 1 hertz, doesn't reach an attenuation (signal reduction) of 100 until about 1.35 hertz. Scaling this to the example, the filter's cutoff frequency must be set to 3.7 kHz so that everything above 5 kHz will have the required attenuation. This results in the frequency band between 3.7 kHz and 5 kHz being wasted on the inadequate roll-off of the analog filter.

A subtle point: the attenuation factor of 100 in this example is probably sufficient even though there are 4096 steps in 12 bits. From Fig. 3-4, 5100 hertz will alias to 4900 hertz, 6000 hertz will alias to 4000 hertz, etc. You don't care what the amplitudes of the signals between 5000 and 6300 hertz are, because they alias into the unusable region between 3700 hertz and 5000 hertz. In order for a frequency to alias into the filter's passband (0 to 3.7 kHz), it must be greater than 6300 hertz, or 1.7 times the filter's cutoff frequency of 3700 hertz. As shown in Fig. 3-11c, the attenuation provided by an 8 pole Chebyshev filter at 1.7 times the cutoff frequency is about 1300, much more adequate than the 100 we started the analysis with. The moral to this story: *In most systems, the frequency band between about 0.4 and 0.5 of the sampling frequency is an unusable wasteland of filter roll-off and aliased signals*. This is a direct result of the limitations of analog filters.

The frequency response of the perfect low-pass filter is *flat* across the entire passband. All of the filters look great in this respect in Fig. 3-11, but only because the vertical axis is displayed on a *logarithmic* scale. Another story is told when the graphs are converted to a *linear* vertical scale, as is shown

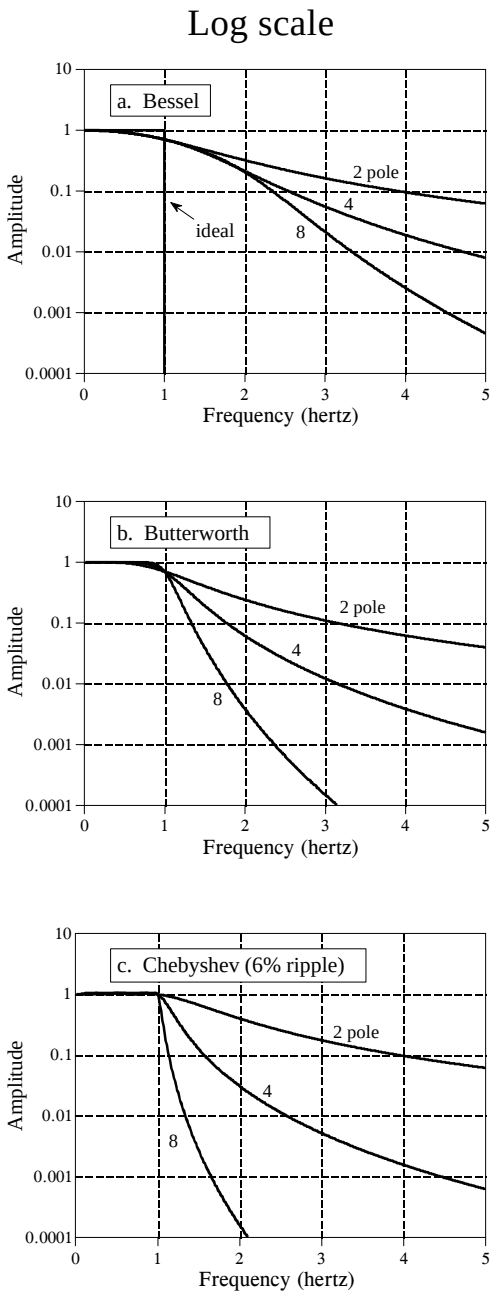


FIGURE 3-11
Frequency response of the three filters on a *logarithmic* scale. The Chebyshev filter has the sharpest roll-off.

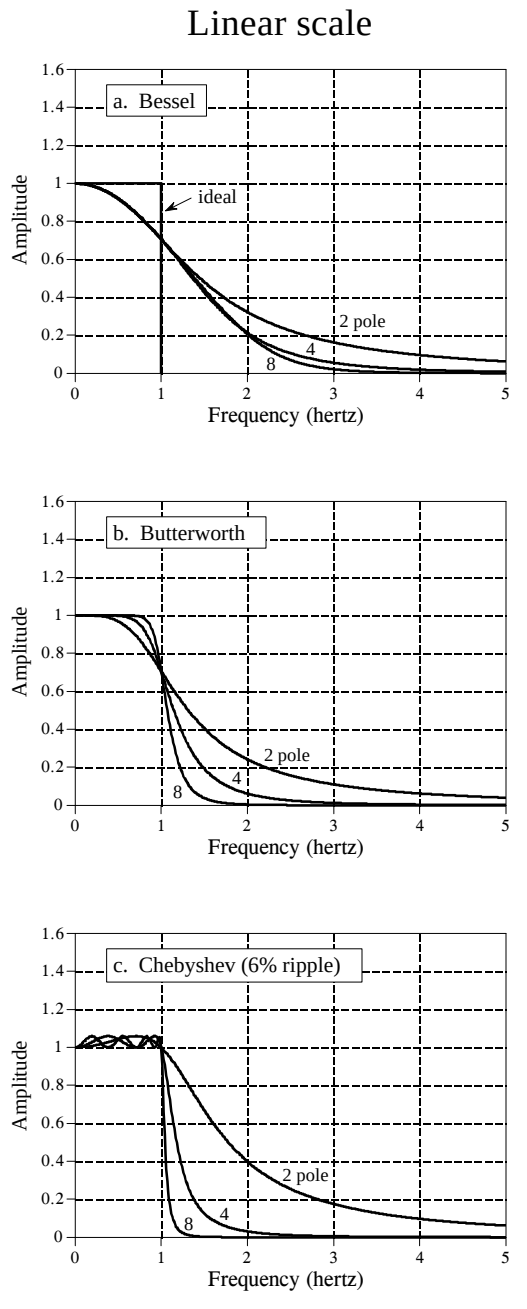


FIGURE 3-12
Frequency response of the three filters on a *linear* scale. The Butterworth filter provides the flattest passband.

in Fig. 3-12. **Passband ripple** can now be seen in the Chebyshev filter (wavy variations in the amplitude of the passed frequencies). In fact, the Chebyshev filter obtains its excellent roll-off by *allowing* this passband ripple. When more passband ripple is allowed in a filter, a faster roll-off

can be achieved. All the Chebyshev filters designed by using Table 3-1 have a passband ripple of about 6% (0.5 dB), a good compromise, and a common choice. A similar design, the **elliptic filter**, allows ripple in both the passband *and* the stopband. Although harder to design, elliptic filters can achieve an even better tradeoff between roll-off and passband ripple.

In comparison, the Butterworth filter is optimized to provide the sharpest roll-off possible *without* allowing ripple in the passband. It is commonly called the *maximally flat filter*, and is identical to a Chebyshev designed for zero passband ripple. The Bessel filter has no ripple in the passband, but the roll-off is far worse than the Butterworth.

The last parameter to evaluate is the **step response**, how the filter responds when the input rapidly changes from one value to another. Figure 3-13 shows the step response of each of the three filters. The horizontal axis is shown for filters with a 1 hertz cutoff frequency, but can be scaled (inversely) for higher cutoff frequencies. For example, a 1000 hertz cutoff frequency would show a step response in *milliseconds*, rather than *seconds*. The Butterworth and Chebyshev filters **overshoot** and show **ringing** (oscillations that slowly decreasing in amplitude). In comparison, the Bessel filter has neither of these nasty problems.

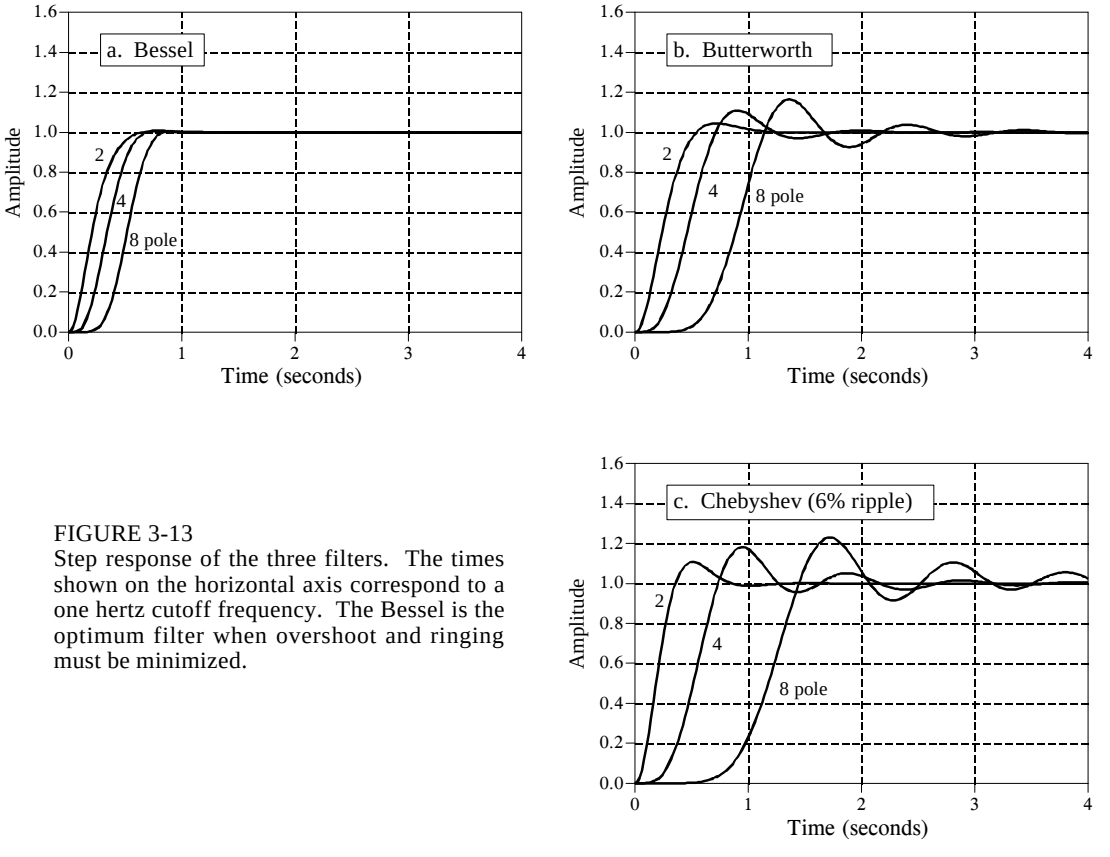


FIGURE 3-13
Step response of the three filters. The times shown on the horizontal axis correspond to a one hertz cutoff frequency. The Bessel is the optimum filter when overshoot and ringing must be minimized.

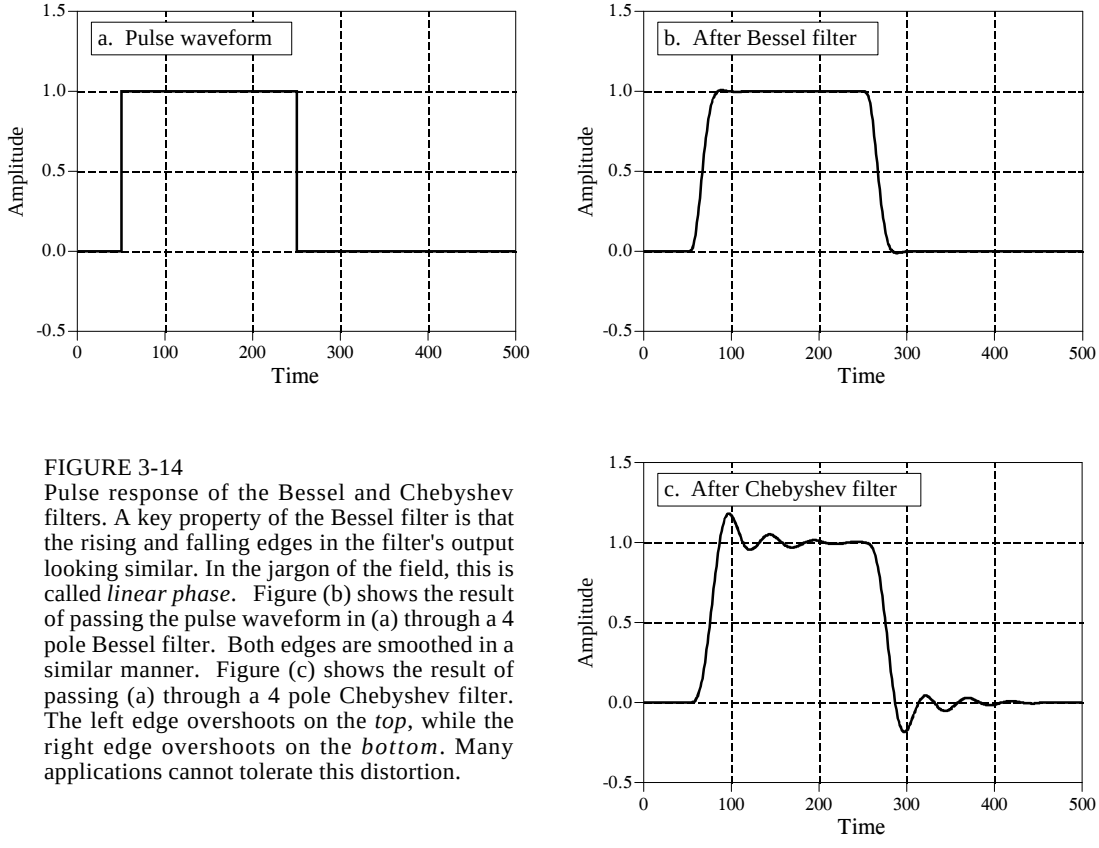


FIGURE 3-14
Pulse response of the Bessel and Chebyshev filters. A key property of the Bessel filter is that the rising and falling edges in the filter's output looking similar. In the jargon of the field, this is called *linear phase*. Figure (b) shows the result of passing the pulse waveform in (a) through a 4 pole Bessel filter. Both edges are smoothed in a similar manner. Figure (c) shows the result of passing (a) through a 4 pole Chebyshev filter. The left edge overshoots on the *top*, while the right edge overshoots on the *bottom*. Many applications cannot tolerate this distortion.

Figure 3-14 further illustrates this very favorable characteristic of the Bessel filter. Figure (a) shows a pulse waveform, which can be viewed as a rising step followed by a falling step. Figures (b) and (c) show how this waveform would appear after Bessel and Chebyshev filters, respectively. If this were a video signal, for instance, the distortion introduced by the Chebyshev filter would be devastating! The overshoot would change the brightness of the *edges* of objects compared to their *centers*. Worse yet, the left side of objects would look bright, while the right side of objects would look dark. Many applications cannot tolerate poor performance in the step response. This is where the Bessel filter shines; no overshoot and symmetrical edges.

Selecting The Antialias Filter

Table 3-2 summarizes the characteristics of these three filters, showing how each optimizes a particular parameter at the expense of everything else. The Chebyshev optimizes the *roll-off*, the Butterworth optimizes the *passband flatness*, and the Bessel optimizes the *step response*.

The selection of the antialias filter depends almost entirely on one issue: *how information is represented in the signals you intend to process*. While

		Step Response			Frequency Response		
	Voltage gain at DC	Overshoot	Time to settle to 1%	Time to settle to 0.1%	Ripple in passband	Frequency for x100 attenuation	Frequency for x1000 attenuation
Bessel							
2 pole	1.27	0.4%	0.60	1.12	0%	12.74	40.4
4 pole	1.91	0.9%	0.66	1.20	0%	4.74	8.45
6 pole	2.87	0.7%	0.74	1.18	0%	3.65	5.43
8 pole	4.32	0.4%	0.80	1.16	0%	3.35	4.53
Butterworth							
2 pole	1.59	4.3%	1.06	1.66	0%	10.0	31.6
4 pole	2.58	10.9%	1.68	2.74	0%	3.17	5.62
6 pole	4.21	14.3%	2.74	3.92	0%	2.16	3.17
8 pole	6.84	16.4%	3.50	5.12	0%	1.78	2.38
Chebyshev							
2 pole	1.84	10.8%	1.10	1.62	6%	12.33	38.9
4 pole	4.21	18.2%	3.04	5.42	6%	2.59	4.47
6 pole	10.71	21.3%	5.86	10.4	6%	1.63	2.26
8 pole	28.58	23.0%	8.34	16.4	6%	1.34	1.66

TABLE 3-2
Characteristics of the three classic filters. The Bessel filter provides the best step response, making it the choice for time domain encoded signals. The Chebyshev and Butterworth filters are used to eliminate frequencies in the stopband, making them ideal for frequency domain encoded signals. Values in this table are in the units of *seconds* and *hertz*, for a one hertz cutoff frequency.

there are many ways for information to be encoded in an analog waveform, only two methods are common, **time domain encoding**, and **frequency domain encoding**. The difference between these two is critical in DSP, and will be a reoccurring theme throughout this book.

In *frequency domain encoding*, the information is contained in *sinusoidal waves* that combine to form the signal. Audio signals are an excellent example of this. When a person hears speech or music, the perceived sound depends on the frequencies present, and not on the particular *shape* of the waveform. This can be shown by passing an audio signal through a circuit that changes the phase of the various sinusoids, but retains their frequency and amplitude. The resulting signal *looks* completely different on an oscilloscope, but *sounds* identical. The pertinent information has been left intact, even though the waveform has been significantly altered. Since aliasing misplaces and overlaps frequency components, it directly destroys information encoded in the frequency domain. Consequently, digitization of these signals usually involves an antialias filter with a sharp cutoff, such as a Chebyshev, Elliptic, or Butterworth. What about the nasty step response of these filters? It doesn't matter; the encoded information isn't affected by this type of distortion.

In contrast, *time domain encoding* uses the *shape of the waveform* to store information. For example, physicians can monitor the electrical activity of a

person's heart by attaching electrodes to their chest and arms (an electrocardiogram or EKG). The *shape* of the EKG waveform provides the information being sought, such as when the various chambers contract during a heartbeat. Images are another example of this type of signal. Rather than a waveform that varies over *time*, images encode information in the shape of a waveform that varies over *distance*. Pictures are formed from regions of brightness and color, and how they relate to other regions of brightness and color. You don't look at the *Mona Lisa* and say, *"My, what an interesting collection of sinusoids."*

Here's the problem: The sampling theorem is an analysis of what happens in the frequency domain during digitization. This makes it ideal to under-stand the analog-to-digital conversion of signals having their information encoded in the frequency domain. However, the sampling theorem is little help in understanding how time domain encoded signals should be digitized. Let's take a closer look.

Figure 3-15 illustrates the choices for digitizing a time domain encoded signal. Figure (a) is an example analog signal to be digitized. In this case, the information we want to capture is the *shape* of the rectangular pulses. A short burst of a high frequency sine wave is also included in this example signal. This represents wideband noise, interference, and similar junk that always appears on analog signals. The other figures show how the digitized signal would appear with different antialias filter options: a Chebyshev filter, a Bessel filter, and no filter.

It is important to understand that *none* of these options will allow the original signal to be reconstructed from the sampled data. This is because the original signal inherently contains frequency components greater than one-half of the sampling rate. Since these frequencies cannot exist in the digitized signal, the reconstructed signal cannot contain them either. These high frequencies result from two sources: (1) noise and interference, which you would like to eliminate, and (2) sharp edges in the waveform, which probably contain information you want to retain.

The Chebyshev filter, shown in (b), attacks the problem by aggressively removing all high frequency components. This results in a filtered analog signal that *can* be sampled and later perfectly reconstructed. However, the reconstructed analog signal is identical to the *filtered signal*, not the *original signal*. Although nothing is lost in sampling, the waveform has been severely distorted by the antialias filter. As shown in (b), the cure is worse than the disease! Don't do it!

The Bessel filter, (c), is designed for just this problem. Its output closely resembles the original waveform, with only a gentle rounding of the edges. By adjusting the filter's cutoff frequency, the smoothness of the edges can be traded for elimination of high frequency components in the signal. Using more poles in the filter allows a *better* tradeoff between these two parameters. A common guideline is to set the cutoff frequency at about one-quarter of the sampling frequency. This results in about two samples

along the rising portion of each edge. Notice that both the Bessel and the Chebyshev filter have removed the burst of high frequency noise present in the original signal.

The last choice is to use no antialias filter at all, as is shown in (d). This has the strong advantage that the value of each sample is *identical* to the value of the original analog signal. In other words, it has perfect edge sharpness; a change in the original signal is immediately mirrored in the digital data. The disadvantage is that aliasing can distort the signal. This takes two different forms. First, high frequency interference and noise, such as the example sinusoidal burst, will turn into meaningless samples, as shown in (d). That is, any high frequency noise present in the analog signal will appear as aliased noise in the digital signal. In a more general sense, this is not a problem of the sampling, but a problem of the upstream analog electronics. It is not the ADC's purpose to reduce noise and interference; this is the responsibility of the analog electronics before the digitization takes place. It may turn out that a Bessel filter should be placed before the digitizer to control this problem. However, this means the filter should be viewed as part of the analog processing, not something that is being done for the sake of the digitizer.

The second manifestation of aliasing is more subtle. When an event occurs in the analog signal (such as an edge), the digital signal in (d) detects the change on the *next* sample. There is no information in the digital data to indicate what happens *between* samples. Now, compare using *no filter* with using a *Bessel filter* for this problem. For example, imagine drawing straight lines between the samples in (c). The time when this constructed line crosses one-half the amplitude of the step provides a *subsample* estimate of when the edge occurred in the analog signal. When no filter is used, this subsample information is completely lost. You don't need a fancy theorem to evaluate how this will affect your particular situation, just a good understanding of what you plan to do with the data once is it acquired.

Multirate Data Conversion

There is a strong trend in electronics to replace *analog circuitry* with *digital algorithms*. Data conversion is an excellent example of this. Consider the design of a digital voice recorder, a system that will digitize a voice signal, store the data in digital form, and later reconstruct the signal for playback. To recreate intelligible speech, the system must capture the frequencies between about 100 and 3000 hertz. However, the analog signal produced by the microphone also contains much higher frequencies, say to 40 kHz. The brute force approach is to pass the analog signal through an eight pole low-pass Chebyshev filter at 3 kHz, and then sample at 8 kHz. On the other end, the DAC reconstructs the analog signal at 8 kHz with a zeroth order hold. Another Chebyshev filter at 3 kHz is used to produce the final voice signal.

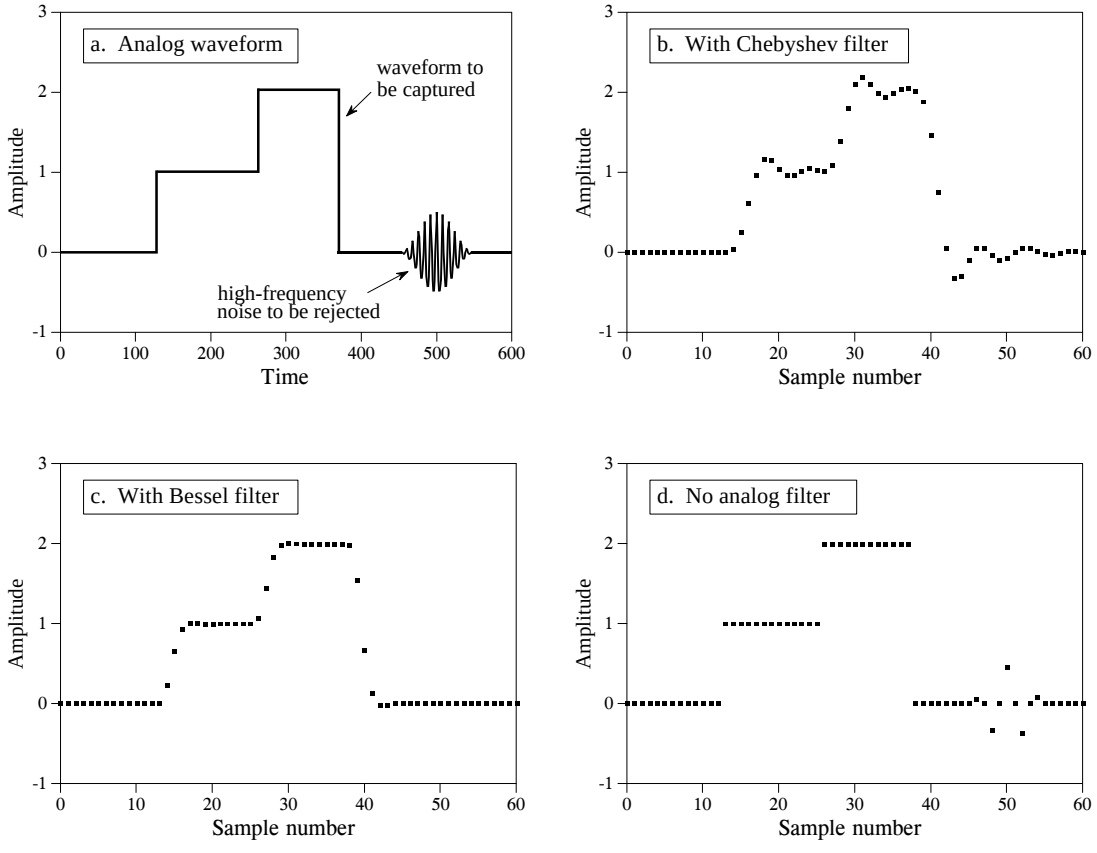


FIGURE 3-15
Three antialias filter options for time domain encoded signals. The goal is to eliminate high frequencies (that will alias during sampling), while simultaneously retaining edge sharpness (that carries information). Figure (a) shows an example analog signal containing both sharp edges and a high frequency noise burst. Figure (b) shows the digitized signal using a *Chebyshev filter*. While the high frequencies have been effectively removed, the edges have been grossly distorted. This is usually a terrible solution. The *Bessel filter*, shown in (c), provides a gentle edge smoothing while removing the high frequencies. Figure (d) shows the digitized signal using *no antialias filter*. In this case, the edges have retained perfect sharpness; however, the high frequency burst has aliased into several meaningless samples.

There are many useful benefits in sampling *faster* than this direct analysis. For example, imagine redesigning the digital voice recorder using a 64 kHz sampling rate. The antialias filter now has an easier task: pass all freq-uencies below 3 kHz, while rejecting all frequencies above 32 kHz. A similar simplification occurs for the reconstruction filter. In short, the higher sampling rate allows the eight pole filters to be replaced with simple resistor-capacitor (RC) networks. The problem is, the digital system is now swamped with data from the higher sampling rate.

The next level of sophistication involves **multirate** techniques, using more than one sampling rate in the same system. It works like this for the digital voice recorder example. First, pass the voice signal through a simple RC low-

pass filter and sample the data at 64 kHz. The resulting digital data contains the desired voice band between 100 and 3000 hertz, but also has an unusable band between 3 kHz and 32 kHz. Second, remove these unusable frequencies in *software*, by using a *digital* low-pass filter at 3 kHz. Third, resample the digital signal from 64 kHz to 8 kHz by simply discarding every seven out of eight samples, a procedure called **decimation.** The resulting digital data is equivalent to that produced by aggressive analog filtering and direct 8 kHz sampling.

Multirate techniques can also be used in the output portion of our example system. The 8 kHz data is pulled from memory and converted to a 64 kHz sampling rate, a procedure called **interpolation**. This involves placing seven samples, with a value of zero, between each of the samples obtained from memory. The resulting signal is a digital *impulse train*, containing the desired voice band between 100 and 3000 hertz, plus spectral duplications between 3 kHz and 32 kHz. Refer back to Figs. 3-6 a&b to understand why this it true. Everything above 3 kHz is then removed with a *digital* low-pass filter. After conversion to an analog signal through a DAC, a simple RC network is all that is required to produce the final voice signal.

Multirate data conversion is valuable for two reasons: (1) it replaces analog components with software, a clear economic advantage in mass-produced products, and (2) it can achieve higher levels of performance in critical applications. For example, compact disc audio systems use techniques of this type to achieve the best possible sound quality. This increased performance is a result of replacing analog components (1% precision), with digital algorithms (0.0001% precision from round-off error). As discussed in upcoming chapters, digital filters outperform analog filters by *hundreds of times* in key areas.

Single Bit Data Conversion

A popular technique in telecommunications and high fidelity music reproduction is **single bit ADC and DAC**. These are multirate techniques where a higher sampling rate is traded for a lower number of bits. In the extreme, only a single bit is needed for each sample. While there are many different circuit configurations, most are based on the use of **delta modulation**. Three example circuits will be presented to give you a flavor of the field. All of these circuits are implemented in IC's, so don't worry where all of the individual transistors and op amps should go. No one is going to ask you to build one of these circuits from basic components.

Figure 3-16 shows the block diagram of a typical delta modulator. The analog input is a voice signal with an amplitude of a few volts, while the output signal is a stream of digital ones and zeros. A comparator decides which has the greater voltage, the incoming analog signal, or the voltage stored on the capacitor. This decision, in the form of a digital one or zero, is applied to the input of the latch. At each clock pulse, typically at a few hundred kilohertz, the latch transfers whatever digital state appears on its

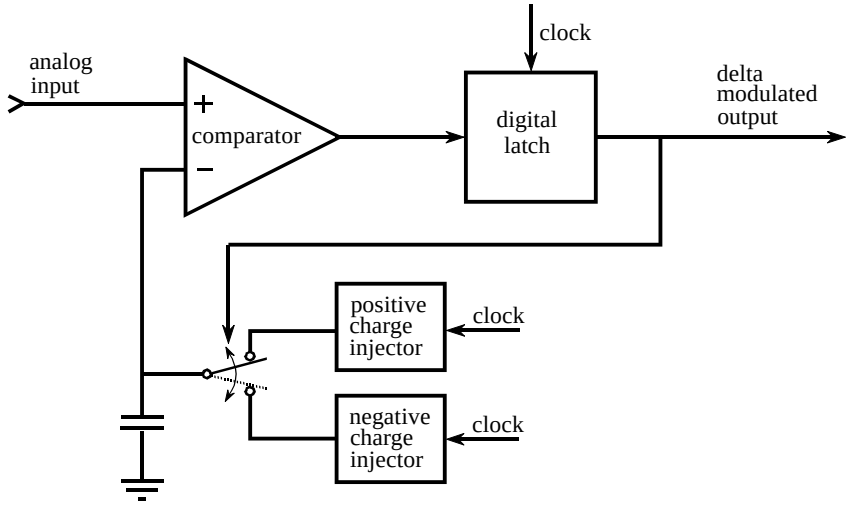


FIGURE 3-16
Block diagram of a delta modulation circuit. The input voltage is compared with the voltage stored on the capacitor, resulting in a digital zero or one being applied to the input of the latch. The output of the latch is updated in synchronization with the clock, and used in a feedback loop to cause the capacitor voltage to track the input voltage.

input, to its output. This latch insures that the output is synchronized with the clock, thereby defining the sampling rate, i.e., the rate at which the 1 bit output can update itself.

A feedback loop is formed by taking the digital output and using it to drive an electronic switch. If the output is a digital *one*, the switch connects the capacitor to a *positive charge injector*. This is a very loose term for a circuit that increases the voltage on the capacitor by a fixed amount, say 1 millivolt per clock cycle. This may be nothing more than a resistor connected to a large positive voltage. If the output is a digital *zero*, the switch is connected to a *negative charge injector*. This *decreases* the voltage on the capacitor by the same fixed amount.

Figure 3-17 illustrates the signals produced by this circuit. At time equal zero, the analog input and the voltage on the capacitor both start with a voltage of zero. As shown in (a), the input signal suddenly increases to 9.5 volts on the eighth clock cycle. Since the input signal is now more positive than the voltage on the capacitor, the digital output changes to a *one*, as shown in (b). This results in the switch being connected to the positive charge injector, and the voltage on the capacitor increasing by a small amount on each clock cycle. Although an increment of 1 volt per clock cycle is shown in (a), this is only for illustration, and a value of 1 millivolt is more typical. This staircase increase in the capacitor voltage continues until it exceeds the voltage of the input signal. Here the system reached an equilibrium with the output oscillating between a digital one and zero, causing the voltage on the capacitor to oscillate between 9 volts and 10

volts. In this manner, the feedback of the circuit forces the capacitor voltage to track the voltage of the input signal. If the input signal changes very rapidly, the voltage on the capacitor changes at a constant rate until a match is obtained. This constant rate of change is called the **slew rate**, just as in other electronic devices such as op amps.

Now, consider the characteristics of the delta modulated output signal. If the analog input is *increasing* in value, the output signal will consist of more ones than zeros. Likewise, if the analog input is *decreasing* in value, the output will consist of more zeros than ones. If the analog input is constant, the digital output will alternate between zero and one with an equal number of each. Put in more general terms, the relative number of ones versus zeros is directly proportional to the *slope* (derivative) of the analog input.

This circuit is a cheap method of transforming an analog signal into a serial stream of ones and zeros for transmission or digital storage. An especially attractive feature is that all the bits have the same meaning, unlike the conventional serial format: *start bit, LSB,⋯ ,MSB, stop bit*. The circuit at the receiver is identical to the feedback portion of the transmitting circuit. Just as the voltage on the capacitor in the transmitting circuit follows the analog input, so does the voltage on the capacitor in the receiving circuit. That is, the capacitor voltage shown in (a) also represents how the reconstructed signal would appear.

A critical limitation of this circuit is the unavoidable tradeoff between (1) maximum slew rate, (2) quantization size, and (3) data rate. In particular, if the maximum slew rate and quantization size are adjusted to acceptable values for voice communication, the data rate ends up in the MHz range. This is too high to be of commercial value. For instance, conventional sampling of a voice signal requires only about 64,000 bits per second.

A solution to this problem is shown in Fig. 3-18, the Continuously Variable Slope Delta (**CVSD**) modulator, a technique implemented in the Motorola MC3518 family. In this approach, the clock rate and the quantization size are set to something acceptable, say 30 kHz, and 2000 levels. This results in a terrible slew rate, which you correct with additional circuitry. In operation, a shift resister continually looks at the last four bits that the system has produced. If the circuit is in a slew rate limited condition, the last four bits will be all ones (positive slope) or all zeros (negative slope). A logic circuit detects this situation and produces an analog signal that increases the level of charge produced by the charge injectors. This boosts the slew rate by increasing the size of the voltage steps being applied to the capacitor.

An analog filter is usually placed between the logic circuitry and the charge injectors. This allows the step size to depend on how long the circuit has been in a slew limited condition. As long as the circuit is slew limited, the step size keeps getting larger and larger. This is often called a *syllabic filter*, since its characteristics depend on the average length of the syllables making up speech. With proper optimization (from the chip manufacturer's

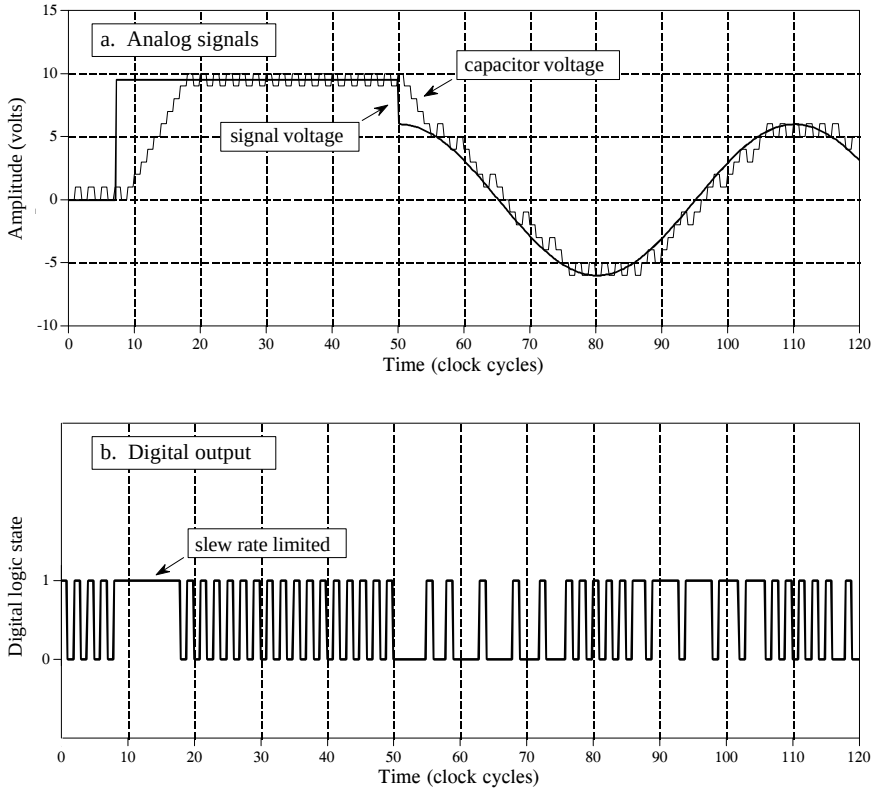


FIGURE 3-17
Example of signals produced by the delta modulator in Fig. 3-16. Figure (a) shows the analog input signal, and the corresponding voltage on the capacitor. Figure (b) shows the delta modulated output, a digital stream of ones and zeros.

spec sheet, not your own work), data rates of 16 to 32 kHz produce acceptable quality speech. The continually changing step size makes the digital data difficult to understand, but fortunately, you don't need to. At the receiver, the analog signal is reconstructed by incorporating a syllabic filter that is identical to the one in the transmission circuit. If the two filters are matched, little distortion results from the CVSD modulation. CVSD is probably the easiest way to digitally transmit a voice signal.

While CVSD modulation is great for encoding voice signals, it cannot be used for general purpose analog-to-digital conversion. Even if you get around the fact that the digital data is related to the *derivative* of the input signal, the *changing step size* will confuse things beyond repair. In addition, the DC level of the analog signal is usually not captured in the digital data.

The **delta-sigma** converter, shown in Fig. 3-19, eliminates these problems by cleverly combining analog electronics with DSP algorithms. Notice that the voltage on the capacitor is now being compared with ground potential. The feedback loop has also been modified so that the voltage on the

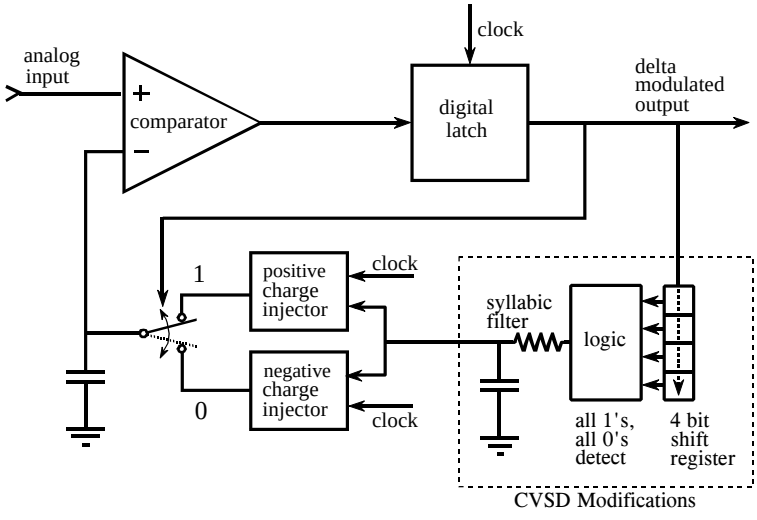


FIGURE 3-18
CVSD modulation block diagram. A logic circuit is added to the basic delta modulator to improve the slew rate.

capacitor is *decreased* when the circuit's output is a digital *one*, and *increased* when it is a digital *zero*. As the input signal increases and decreases in voltage, it tries to raise and lower the voltage on the capacitor. This change in voltage is detected by the comparator, resulting in the charge injectors producing a *counteracting* charge to keep the capacitor at zero volts.

If the input voltage is positive, the digital output will be composed of more ones than zeros. The excess number of ones being needed to generate the *negative* charge that cancels with the *positive* input signal. Likewise, if the input voltage is negative, the digital output will be composed of more zeros than ones, providing a net positive charge injection. If the input signal is equal to zero volts, an equal number of ones and zeros will be generated in the output, providing an overall charge injection of zero.

The relative number of ones and zeros in the output is now related to the *level* of the input voltage, not the *slope* as in the previous circuit. This is much simpler. For instance, you could form a 12 bit ADC by feeding the digital output into a counter, and counting the number of *ones* over 4096 clock cycles. A digital number of 4095 would correspond to the maximum positive input voltage. Likewise, digital number 0 would correspond to the maximum negative input voltage, and 2048 would correspond to an input voltage of zero. This also shows the origin of the name, *delta-sigma*: delta modulation followed by summation (sigma).

The ones and zeros produced by this type of delta modulator are very easy to transform back into an analog signal. All that is required is an analog low-pass filter, which might be as simple as a single RC network. The high

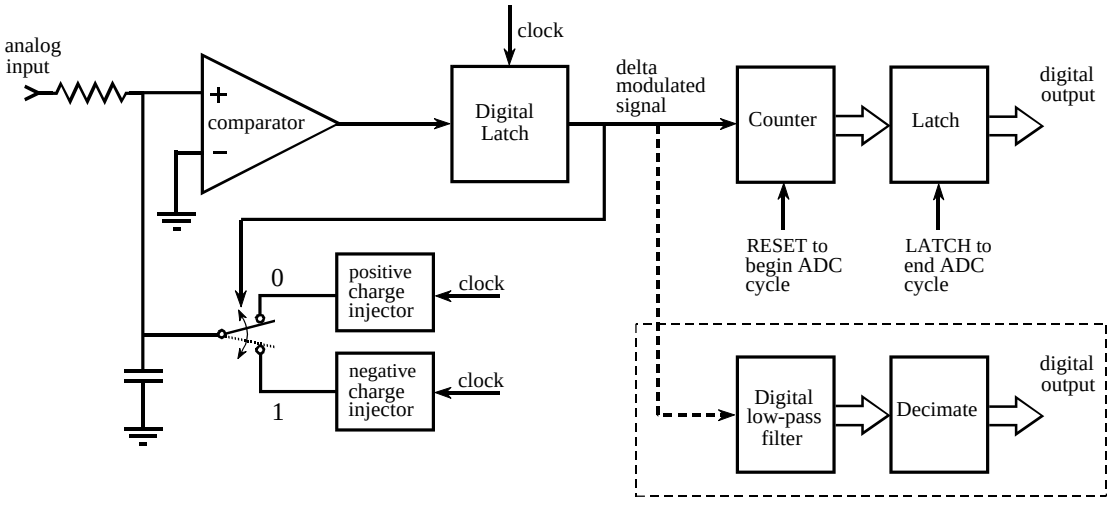


FIGURE 3-19
Block diagram of a delta-sigma analog-to-digital converter. In the simplest case, the pulses from a delta modulator are counted for a predetermined number of clock cycles. The output of the counter is then latched to complete the conversion. In a more sophisticated circuit, the pulses are passed through a digital low-pass filter and then resampled (decimated) to a lower sampling rate.

and low voltages corresponding to the digital ones and zeros average out to form the correct analog voltage. For example, suppose that the ones and zeros are represented by 5 volts and 0 volts, respectively. If 80% of the bits in the data stream are *ones*, and 20% are *zeros*, the output of the low-pass filter will be 4 volts.

This method of transforming the single bit data stream back into the original waveform is important for several reasons. First, it describes a slick way to replace the counter in the delta-sigma ADC circuit. Instead of simply counting the pulses from the delta modulator, the binary signal is passed through a *digital* low-pass filter, and then *decimated* to reduce the sampling rate. For example, this procedure might start by changing each of the ones and zeros in the digital stream into a 12 bit sample; ones become a value of 4095, while zeros become a value of 0. Using a digital low-pass filter on this signal produces a digitized version of the original waveform, just as an analog low-pass filter would form an analog recreation. Decimation then reduces the sampling rate by discarding most of the samples. This results in a digital signal that is equivalent to direct sampling of the original waveform.

This approach is used in many commercial ADC's for digitizing voice and other audio signals. An example is the National Semiconductor ADC16071, which provides 16 bit analog-to-digital conversion at sampling rates up to 192 kHz. At a sampling rate of 100 kHz, the delta modulator operates with a clock frequency of 6.4 MHz. The low-pass digital filter is a 246 point FIR, such as described in Chapter 16. This removes all frequencies in the digital data above 50 kHz, ½ of the eventual sampling rate. Conceptually, this can be

viewed as forming a digital signal at 6.4 MHz, with each sample represented by 16 bits. The signal is then decimated from 6.4 MHz to 100 kHz, accomplished by deleting every 63 out of 64 samples. In actual operation, much more goes on inside of this device than described by this simple discussion.

Delta-sigma converters can also be used for digital-to-analog conversion of voice and audio signals. The digital signal is retrieved from memory, and converted into a delta modulated stream of ones and zeros. As mentioned above, this single bit signal can easily be changed into the reconstructed analog signal with a simple low-pass analog filter. As with the antialias filter, usually only a single RC network is required. This is because the majority of the filtration is handled by the high-performance digital filters.

Delta-sigma ADC's have several quirks that limit their use to specific applications. For example, it is difficult to multiplex their inputs. When the input is switched from one signal to another, proper operation is not established until the digital filter can clear itself of data from the previous signal. Delta-sigma converters are also limited in another respect: you don't know exactly *when* each sample was taken. Each acquired sample is a composite of the one bit information taken over a segment of the input signal. This is not a problem for signals encoded in the frequency domain, such as audio, but it is a significant limitation for time domain encoded signals. To understand the shape of a signal's waveform, you often need to know the precise instant each sample was taken. Lastly, most of these devices are specifically designed for audio applications, and their performance specifications are quoted accordingly. For example, a 16 bit ADC used for voice signals does not necessarily mean that each sample has 16 bits of precision. Much more likely, the manufacturer is stating that *voice signals* can be digitized to 16 bits of *dynamic range*. Don't expect to get a full 16 bits of useful information from this device for general purpose data acquisition.

While these explanations and examples provide an introduction to single bit ADC and DAC, it must be emphasized that they are simplified descriptions of sophisticated DSP and integrated circuit technology. You wouldn't expect the manufacturer to tell their *competitors* all the internal workings of their chips, so don't expect them to tell *you*.