CHAPTER 9

Applications of the DFT

The Discrete Fourier Transform (DFT) is one of the most important tools in Digital Signal Processing. This chapter discusses three common ways it is used. First, the DFT can calculate a signal's *frequency spectrum*. This is a direct examination of information encoded in the frequency, phase, and amplitude of the component sinusoids. For example, human speech and hearing use signals with this type of encoding. Second, the DFT can find a system's frequency response from the system's impulse response, and vice versa. This allows systems to be analyzed in the *frequency domain*, just as convolution allows systems to be analyzed in the *time domain*. Third, the DFT can be used as an intermediate step in more elaborate signal processing techniques. The classic example of this is *FFT convolution*, an algorithm for convolving signals that is hundreds of times faster than conventional methods.

Spectral Analysis of Signals

It is very common for information to be encoded in the sinusoids that form a signal. This is true of naturally occurring signals, as well as those that have been created by humans. Many things oscillate in our universe. For example, speech is a result of vibration of the human vocal cords; stars and planets change their brightness as they rotate on their axes and revolve around each other; ship's propellers generate periodic displacement of the water, and so on. The *shape* of the time domain waveform is not important in these signals; the key information is in the *frequency*, *phase* and *amplitude* of the component sinusoids. The DFT is used to extract this information.

An example will show how this works. Suppose we want to investigate the sounds that travel through the ocean. To begin, a microphone is placed in the water and the resulting electronic signal amplified to a reasonable level, say a few volts. An analog low-pass filter is then used to remove all frequencies above 80 hertz, so that the signal can be digitized at 160 samples per second. After acquiring and storing several thousand samples, what next?

The first thing is to simply *look* at the data. Figure 9-1a shows 256 samples from our imaginary experiment. All that can be seen is a noisy waveform that conveys little information to the human eye. For reasons explained shortly, the next step is to multiply this signal by a smooth curve called a **Hamming window**, shown in (b). (Chapter 16 provides the equations for the Hamming and other windows; see Eqs. 16-1 and 16-2, and Fig. 16-2a). This results in a 256 point signal where the samples near the ends have been reduced in amplitude, as shown in (c).

Taking the DFT, and converting to polar notation, results in the 129 point frequency spectrum in (d). Unfortunately, this also looks like a noisy mess. This is because there is not enough information in the original 256 points to obtain a well behaved curve. Using a longer DFT does nothing to help this problem. For example, if a 2048 point DFT is used, the frequency spectrum becomes 1025 samples long. Even though the original 2048 points contain more information, the greater number of samples in the spectrum dilutes the information by the same factor. Longer DFTs provide better frequency resolution, but the same noise level.

The answer is to use more of the original signal in a way that doesn't increase the number of points in the frequency spectrum. This can be done by breaking the input signal into many 256 point *segments*. Each of these segments is multiplied by the Hamming window, run through a 256 point DFT, and converted to polar notation. The resulting frequency spectra are then *averaged* to form a single 129 point frequency spectrum. Figure (e) shows an example of averaging 100 of the frequency spectra typified by (d). The improvement is obvious; the noise has been reduced to a level that allows interesting features of the signal to be observed. Only the *magnitude* of the frequency domain is averaged in this manner; the *phase* is usually discarded because it doesn't contain useful information. The random noise reduces in proportion to the *square-root* of the number of segments. While 100 segments is typical, some applications might average *millions* of segments to bring out weak features.

There is also a second method for reducing spectral noise. Start by taking a very long DFT, say 16,384 points. The resulting frequency spectrum is high resolution (8193 samples), but very noisy. A low-pass digital filter is then used to *smooth* the spectrum, reducing the noise at the expense of the resolution. For example, the simplest digital filter might average 64 adjacent samples in the original spectrum to produce each sample in the filtered spectrum. Going through the calculations, this provides about the same noise and resolution as the first method, where the 16,384 points would be broken into 64 segments of 256 points each.

Which method should you use? The first method is easier, because the digital filter isn't needed. The second method has the *potential* of better performance, because the digital filter can be tailored to optimize the trade-off between noise and resolution. However, this improved performance is seldom worth the trouble. This is because both noise and resolution can be improved by using *more data* from the input signal. For example,

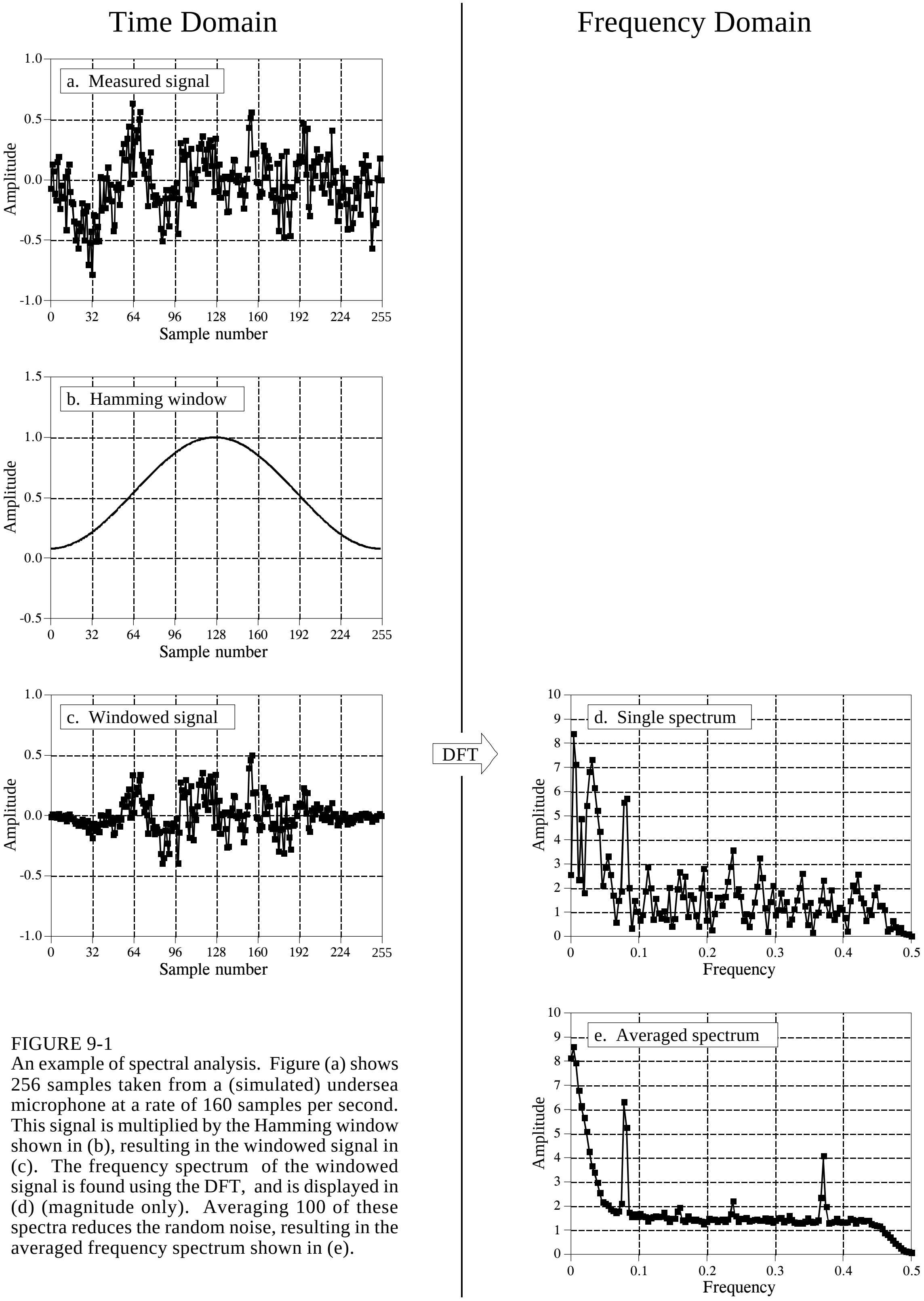

FIGURE 9-1
An example of spectral analysis. Figure (a) shows 256 samples taken from a (simulated) undersea microphone at a rate of 160 samples per second. This signal is multiplied by the Hamming window shown in (b), resulting in the windowed signal in (c). The frequency spectrum of the windowed signal is found using the DFT, and is displayed in (d) (magnitude only). Averaging 100 of these spectra reduces the random noise, resulting in the averaged frequency spectrum shown in (e).

imagine breaking the acquired data into 10,000 segments of 16,384 samples each. This resulting frequency spectrum is high resolution (8193 points) *and* low noise (10,000 averages). Problem solved! For this reason, we will only look at the averaged segment method in this discussion.

Figure 9-2 shows an example spectrum from our undersea microphone, illustrating the features that commonly appear in the frequency spectra of acquired signals. Ignore the sharp peaks for a moment. Between 10 and 70 hertz, the signal consists of a relatively flat region. This is called **white noise** because it contains an equal amount of all frequencies, the same as white light. It results from the noise on the time domain waveform being *uncorrelated* from sample-to-sample. That is, knowing the noise value present on any one sample provides no information on the noise value present on any other sample. For example, the random motion of electrons in electronic circuits produces white noise. As a more familiar example, the sound of the water spray hitting the shower floor is white noise. The white noise shown in Fig. 9-2 could be originating from any of several sources, including the analog electronics, or the ocean itself.

Above 70 hertz, the white noise rapidly decreases in amplitude. This is a result of the roll-off of the antialias filter. An ideal filter would pass all frequencies below 80 hertz, and block all frequencies above. In practice, a perfectly sharp cutoff isn't possible, and you should expect to see this gradual drop. If you don't, suspect that an aliasing problem is present.

Below about 10 hertz, the noise rapidly increases due to a curiosity called **1/f noise** (one-over-f noise). 1/f noise is a mystery. It has been measured in very diverse systems, such as traffic density on freeways and electronic noise in transistors. It probably could be measured in all systems, if you look low enough in frequency. In spite of its wide occurrence, a general theory and understanding of 1/f noise has eluded researchers. The cause of this noise can be identified in some specific systems; however, this doesn't answer the question of why 1/f noise is everywhere. For common analog electronics and most physical systems, the transition between white noise and 1/f noise occurs between about 1 and 100 hertz.

Now we come to the sharp peaks in Fig. 9-2. The easiest to explain is at 60 hertz, a result of electromagnetic interference from commercial electrical power. Also expect to see smaller peaks at multiples of this frequency (120, 180, 240 hertz, etc.) since the power line waveform is not a *perfect* sinusoid. It is also common to find interfering peaks between 25-40 kHz, a favorite for designers of switching power supplies. Nearby radio and television stations produce interfering peaks in the megahertz range. Low frequency peaks can be caused by components in the system vibrating when shaken. This is called *microphonics*, and typically creates peaks at 10 to 100 hertz.

Now we come to the actual signals. There is a strong peak at 13 hertz, with weaker peaks at 26 and 39 hertz. As discussed in the next chapter, this is the frequency spectrum of a nonsinusoidal periodic waveform. The peak at 13 hertz is called the fundamental frequency, while the peaks at 26 and 39

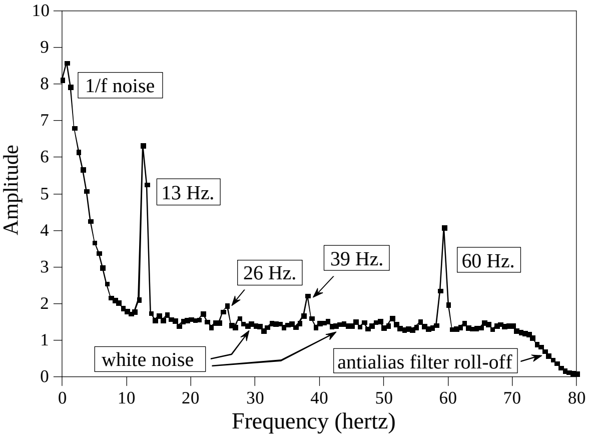

FIGURE 9-2
Example frequency spectrum. Three types of features appear in the spectra of acquired signals: (1) random noise, such as white noise and 1/f noise, (2) interfering signals from power lines, switching power supplies, radio and TV stations, microphonics, etc., and (3) real signals, usually appearing as a fundamental plus harmonics. This example spectrum (magnitude only) shows several of these features.

hertz are referred to as the second and third harmonic respectively. You would also expect to find peaks at other multiples of 13 hertz, such as 52, 65, 78 hertz, etc. You don't see these in Fig. 9-2 because they are buried in the white noise. This 13 hertz signal might be generated, for example, by a submarines's three bladed propeller turning at 4.33 revolutions per second. This is the basis of *passive* sonar, identifying undersea sounds by their frequency and harmonic content.

Suppose there are peaks very close together, such as shown in Fig. 9-3. There are two factors that limit the frequency resolution that can be obtained, that is, how close the peaks can be without merging into a single entity. The first factor is the length of the DFT. The frequency spectrum produced by an N point DFT consists of $N/2+1$ samples equally spaced between zero and one-half of the sampling frequency. To separate two closely spaced frequencies, the sample spacing must be *smaller* than the distance between the two peaks. For example, a 512 point DFT is sufficient to separate the peaks in Fig. 9-3, while a 128 point DFT is not.

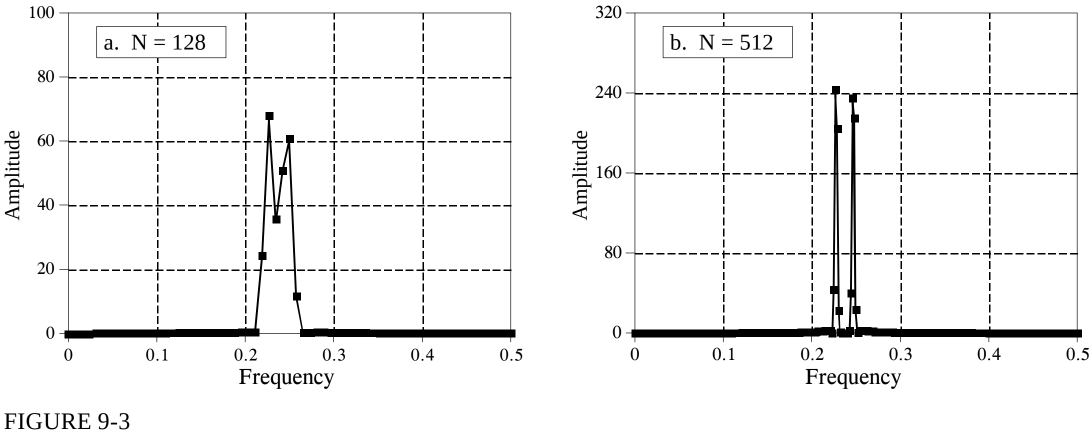

FIGURE 9-3
Frequency spectrum resolution. The longer the DFT, the better the ability to separate closely spaced features. In these example magnitudes, a 128 point DFT cannot resolve the two peaks, while a 512 point DFT can.

The second factor limiting resolution is more subtle. Imagine a signal created by adding two sine waves with only a slight difference in their frequencies. Over a short segment of this signal, say a few periods, the waveform will look like a *single* sine wave. The closer the frequencies, the longer the segment must be to conclude that more than one frequency is present. In other words, the *length* of the signal limits the frequency resolution. This is distinct from the first factor, because the *length of the input signal* does not have to be the same as the *length of the DFT*. For example, a 256 point signal could be padded with zeros to make it 2048 points long. Taking a 2048 point DFT produces a frequency spectrum with 1025 samples. The added zeros don't change the shape of the spectrum, they only provide more samples in the frequency domain. In spite of this very close sampling, the ability to separate closely spaced peaks would be only slightly better than using a 256 point DFT. When the DFT is the same length as the input signal, the resolution is limited about equally by these two factors. We will come back to this issue shortly.

Next question: What happens if the input signal contains a sinusoid with a frequency *between* two of the basis functions? Figure 9-4a shows the answer. This is the frequency spectrum of a signal composed of two sine waves, one having a frequency *matching* a basis function, and the other with a frequency *between* two of the basis functions. As you should expect, the first sine wave is represented as a single point. The other peak is more difficult to understand. Since it cannot be represented by a single sample, it becomes a peak with **tails** that extend a significant distance away.

The solution? Multiply the signal by a Hamming window before taking the DFT, as was previously discussed. Figure (b) shows that the spectrum is changed in three ways by using the window. First, the two peaks are made to look more alike. This is good. Second, the tails are greatly reduced.

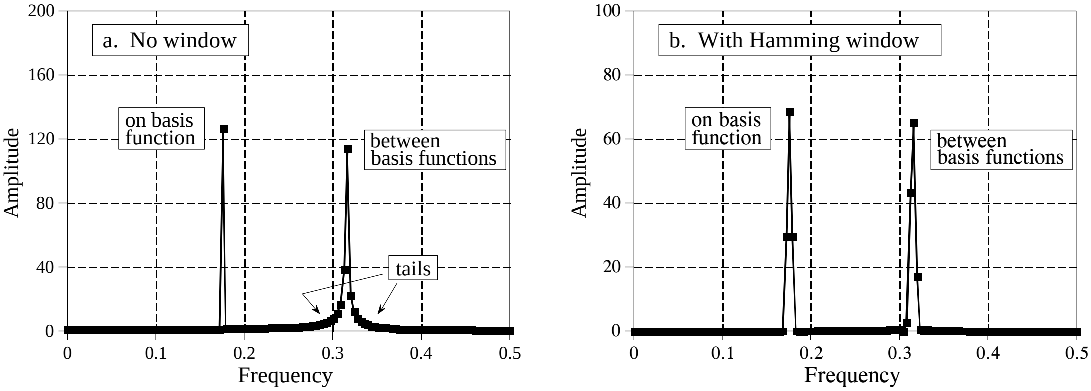

FIGURE 9-4
Example of using a window in spectral analysis. Figure (a) shows the frequency spectrum (magnitude only) of a signal consisting of two sine waves. One sine wave has a frequency exactly equal to a basis function, allowing it to be represented by a single sample. The other sine wave has a frequency *between* two of the basis functions, resulting in *tails* on the peak. Figure (b) shows the frequency spectrum of the same signal, but with a Hamming window applied before taking the DFT. The window makes the peaks look the same and reduces the tails, but broadens the peaks.

This is also good. Third, the window reduces the resolution in the spectrum by making the peaks wider. This is bad. In DSP jargon, windows provide a tradeoff between *resolution* (the width of the peak) and *spectral leakage* (the amplitude of the tails).

To explore the theoretical aspects of this in more detail, imagine an infinitely long discrete sine wave at a frequency of 0.1 the sampling rate. The frequency spectrum of this signal is an infinitesimally narrow peak, with all other frequencies being zero. Of course, neither this signal nor its frequency spectrum can be brought into a digital computer, because of their infinite and infinitesimal nature. To get around this, we change the signal in two ways, both of which distort the true frequency spectrum.

First, we *truncate* the information in the signal, by multiplying it by a window. For example, a 256 point *rectangular window* would allow 256 points to retain their correct value, while all the other samples in the infinitely long signal would be set to a value of zero. Likewise, the Hamming window would *shape* the retained samples, besides setting all points outside the window to zero. The signal is still infinitely long, but only a finite number of the samples have a nonzero value.

How does this windowing affect the frequency domain? As discussed in Chapter 10, when two time domain signals are *multiplied*, the corresponding frequency domains are *convolved*. Since the original spectrum is an infinitesimally narrow peak (i.e., a delta function), the spectrum of the windowed signal is the spectrum of the window shifted to the location of the peak. Figure 9-5 shows how the spectral peak would appear using four different window options (If you need a refresher on dB, look ahead to Chapter 14). Figure 9-5a results from a rectangular window. Figures (b) and (c) result from using two popular windows, the Hamming and the Blackman (as previously mentioned, see Eqs. 16-1 and 16-2, and Fig. 16-2a for information on these windows).

As shown in Fig. 9-5, all these windows have degraded the original spectrum by broadening the peak and adding tails composed of numerous side lobes. This is an unavoidable result of using only a portion of the original time domain signal. Here we can see the tradeoff between the three windows. The Blackman has the widest main lobe (bad), but the lowest amplitude tails (good). The rectangular window has the narrowest main lobe (good) but the largest tails (bad). The Hamming window sits between these two.

Notice in Fig. 9-5 that the frequency spectra are continuous curves, not discrete samples. After windowing, the time domain signal is still infinitely long, even though most of the samples are zero. This means that the frequency spectrum consists of $\infty/2+1$ samples between 0 and 0.5, the same as a continuous line.

This brings in the second way we need to modify the time domain signal to allow it to be represented in a computer: *select N points from the signal*. These N points must contain all the nonzero points identified by the window, but may also include any number of the zeros. This has the effect

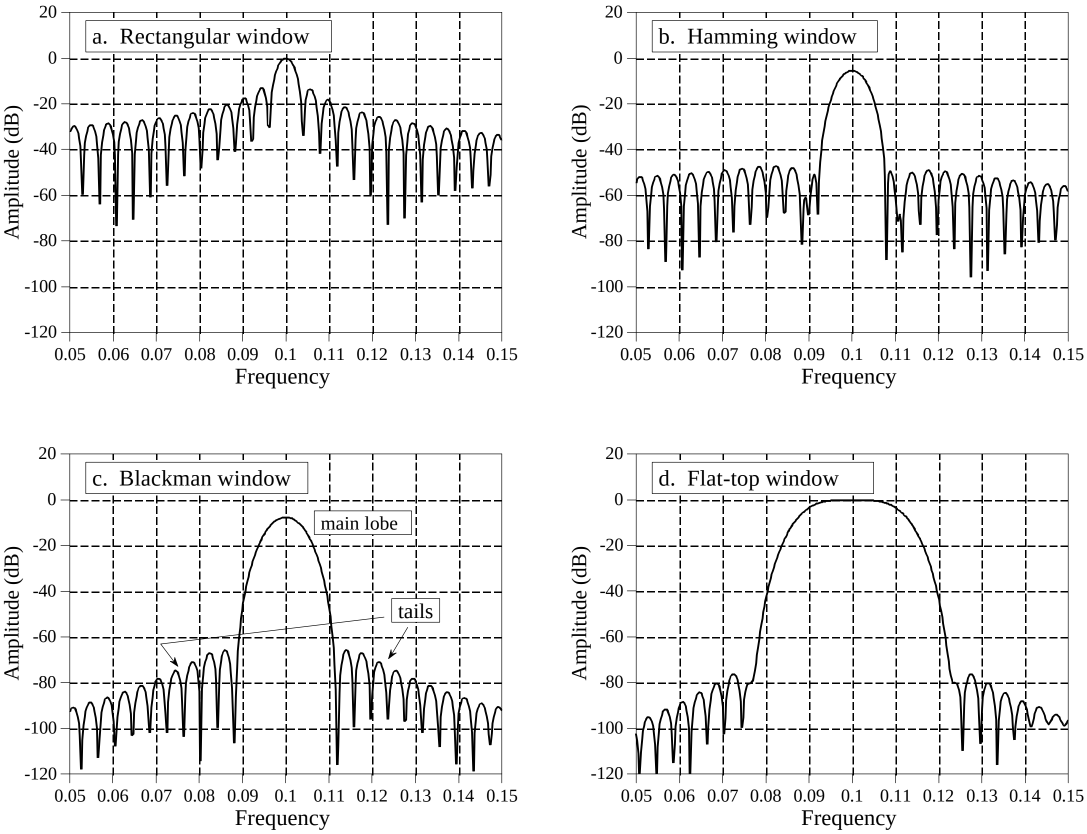

FIGURE 9-5
Detailed view of a spectral peak using various windows. Each peak in the frequency spectrum is a central lobe surrounded by tails formed from side lobes. By changing the window shape, the amplitude of the side lobes can be reduced at the expense of making the main lobe wider. The rectangular window, (a), has the narrowest main lobe but the largest amplitude side lobes. The Hamming window, (b), and the Blackman window, (c), have lower amplitude side lobes at the expense of a wider main lobe. The flat-top window, (d), is used when the amplitude of a peak must be accurately measured. These curves are for 255 point windows; longer windows produce proportionately narrower peaks.

of *sampling* the frequency spectrum's continuous curve. For example, if *N* is chosen to be 1024, the spectrum's continuous curve will be sampled 513 times between 0 and 0.5. If *N* is chosen to be much larger than the window length, the samples in the frequency domain will be close enough that the peaks and valleys of the continuous curve will be preserved in the new spectrum. If *N* is made the same as the window length, the fewer number of samples in the spectrum results in the regular pattern of peaks and valleys turning into irregular tails, depending on where the samples happen to fall. This explains why the two peaks in Fig. 9-4a do not look alike. Each peak in Fig 9-4a is a *sampling* of the underlying curve in Fig. 9-5a. The presence or absence of the tails depends on where the samples are taken in relation to the peaks and valleys. If the sine wave exactly matches a basis function, the samples occur exactly at the valleys, eliminating the tails. If the sine wave is between two basis functions, the samples occur somewhere along the peaks and valleys, resulting in various patterns of tails.

This leads us to the **flat-top window**, shown in Fig. 9-5d. In some applications the *amplitude* of a spectral peak must be measured very accurately. Since the DFT's frequency spectrum is formed from samples, there is nothing to guarantee that a sample will occur exactly at the top of a peak. More than likely, the nearest sample will be slightly off-center, giving a value lower than the true amplitude. The solution is to use a window that produces a spectral peak with a *flat top*, insuring that one or more of the samples will always have the correct peak value. As shown in Fig. 9-5d, the penalty for this is a very broad main lobe, resulting in poor frequency resolution.

As it turns out, the shape we want for a flat-top window is exactly the same shape as the filter kernel of a low-pass filter. We will discuss the theoretical reasons for this in later chapters; for now, here is a cookbook description of how the technique is used. Chapter 16 discusses a low-pass filter called the *windowed-sinc.* Equation 16-4 describes how to generate the filter kernel (which we want to use as a window), and Fig. 16-4a illustrates the typical shape of the curve. To use this equation, you will need to know the value of two parameters: M and f_c. These are found from the relations: $M = N - 2$, and $f_c = s/N$, where N is the length of the DFT being used, and s is the number of samples you want on the flat portion of the peak (usually between 3 and 5). Table 16-1 shows a program for calculating the filter kernel (our window), including two subtle features: the normalization constant, K, and how to avoid a divide-by-zero error on the center sample. When using this method, remember that a DC value of *one* in the time domain will produce a peak of amplitude *one* in the frequency domain. However, a sinusoid of amplitude *one* in the time domain will only produce a spectral peak of amplitude *one-half.* (This is discussed in the last chapter: *Synthesis, Calculating the Inverse DFT*).

Frequency Response of Systems

Systems are analyzed in the *time domain* by using convolution. A similar analysis can be done in the *frequency domain.* Using the Fourier transform, every input signal can be represented as a group of cosine waves, each with a specified amplitude and phase shift. Likewise, the DFT can be used to represent every output signal in a similar form. This means that any linear system can be *completely* described by how it changes the amplitude and phase of cosine waves passing through it. This information is called the system's **frequency response.** Since both the impulse response and the frequency response contain complete information about the system, there must be a one-to-one correspondence between the two. Given one, you can calculate the other. The relationship between the impulse response and the frequency response is one of the foundations of signal processing: *A system's frequency response is the Fourier Transform of its impulse response.* Figure 9-6 illustrates these relationships.

Keeping with standard DSP notation, impulse responses use lower case variables, while the corresponding frequency responses are upper case. Since $h[\ ]$ is the common symbol for the impulse response, $H[\ ]$ is used for the frequency response. Systems are described in the time domain by convolution, that is:

$x[n] \ast h[n] = y[n]$. In the frequency domain, the input spectrum is *multiplied* by the frequency response, resulting in the output spectrum. As an equation: $X[f] \times H[f] = Y[f]$. That is, *convolution* in the time domain corresponds to *multiplication* in the frequency domain.

Figure 9-7 shows an example of using the DFT to convert a system's impulse response into its frequency response. Figure (a) is the impulse response of the system. Looking at this curve isn't going to give you the slightest idea what the system does. Taking a 64 point DFT of this impulse response produces the frequency response of the system, shown in (b). Now the function of this system becomes obvious, it passes frequencies between 0.2 and 0.3, and rejects all others. It is a band-pass filter. The *phase* of the frequency response could also be examined; however, it is *more* difficult to interpret and *less* interesting. It will be discussed in upcoming chapters.

Figure (b) is very jagged due to the low number of samples defining the curve. This situation can be improved by **padding** the impulse response with zeros before taking the DFT. For example, adding zeros to make the impulse response 512 samples long, as shown in (c), results in the higher resolution frequency response shown in (d).

How much resolution can you obtain in the frequency response? The answer is: *infinitely* high, if you are willing to pad the impulse response with an *infinite* number of zeros. In other words, there is nothing limiting the frequency resolution except the length of the DFT. This leads to a very important concept. Even though the impulse response is a *discrete* signal, the corresponding frequency response is *continuous*. An N point DFT of the impulse response provides $N/2 + 1$ *samples* of this continuous curve. If you make the DFT longer, the resolution improves, and you obtain a better idea of

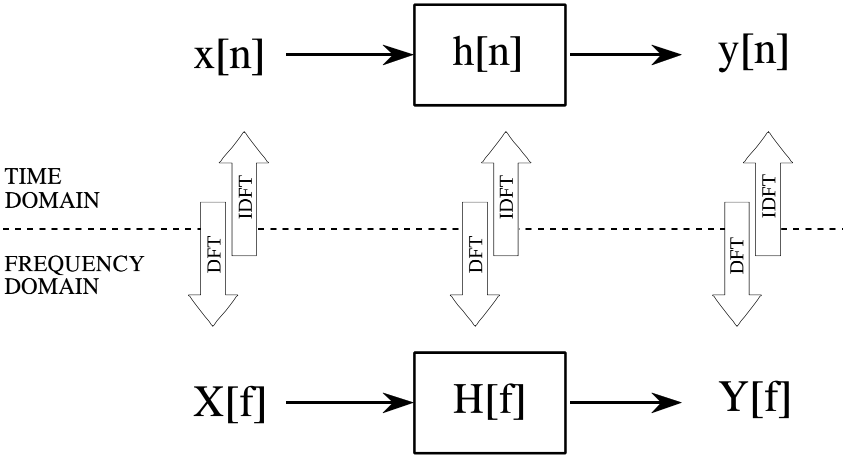

FIGURE 9-6
Comparing system operation in the time and frequency domains. In the time domain, an input signal is *convolved* with an impulse response, resulting in the output signal, that is, $x[n] \ast h[n] = y[n]$. In the frequency domain, an input spectrum is *multiplied* by a frequency response, resulting in the output spectrum, that is, $X[f] \times H[f] = Y[f]$. The DFT and the Inverse DFT relate the signals in the two domain.

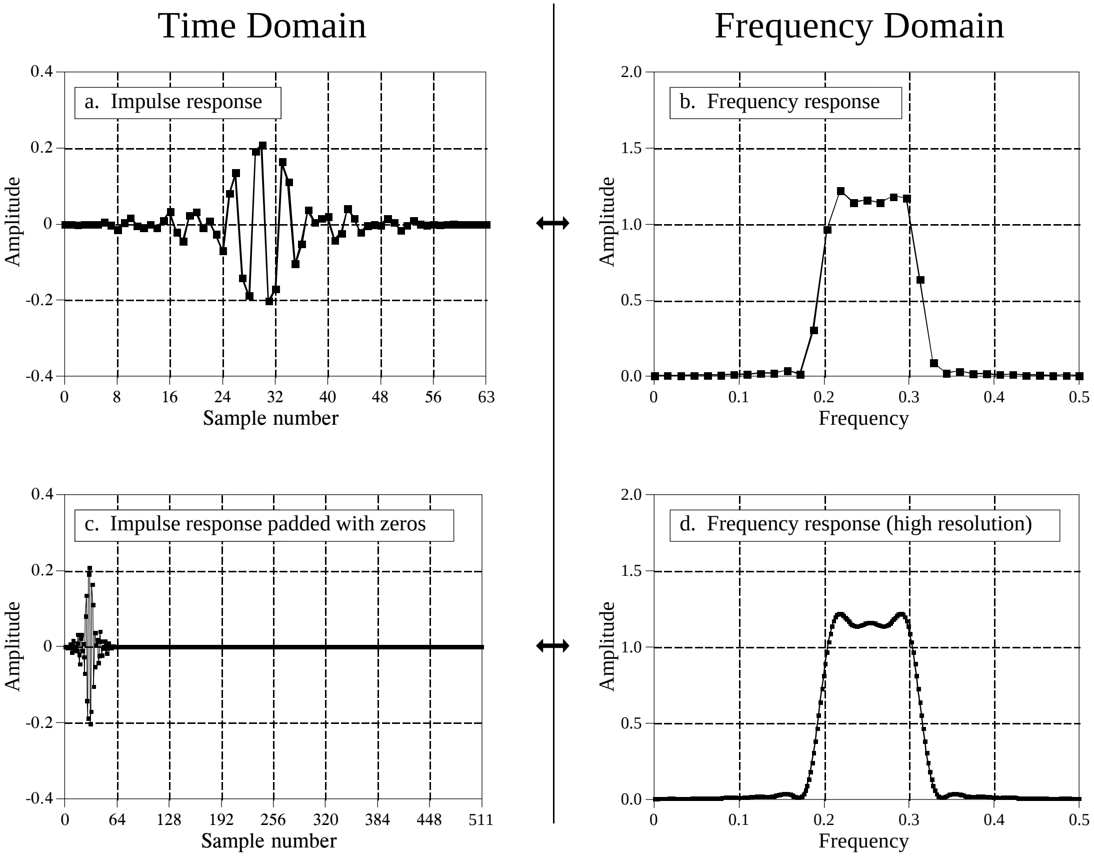

FIGURE 9-7
Finding the frequency response from the impulse response. By using the DFT, a system's impulse response, (a), can be transformed into the system's frequency response, (b). By padding the impulse response with zeros (c), higher resolution can be obtained in the frequency response, (d). Only the magnitude of the frequency response is shown in this example; discussion of the phase is postponed until the next chapter.

what the continuous curve looks like. Remember what the frequency response represents: amplitude and phase changes experienced by cosine waves as they pass through the system. Since the input signal can contain *any* frequency between 0 and 0.5, the system's frequency response *must* be a continuous curve over this range.

This can be better understood by bringing in another member of the Fourier transform family, the **Discrete Time Fourier Transform (DTFT)**. Consider an *N* sample signal being run through an *N* point DFT, producing an $N/2+1$ sample frequency domain. Remember from the last chapter that the DFT considers the time domain signal to be *infinitely long* and *periodic*. That is, the *N* points are repeated over and over from negative to positive infinity. Now consider what happens when we start to pad the time domain signal with an ever increasing number of zeros, to obtain a finer and finer sampling in the frequency domain. Adding zeros makes the period of the time domain *longer*, while simultaneously making the frequency domain samples *closer together*.

Now we will take this to the extreme, by adding an *infinite* number of zeros to the time domain signal. This produces a different situation in two respects. First, the time domain signal now has an infinitely long period. In other words, it has turned into an *aperiodic* signal. Second, the frequency domain has achieved an infinitesimally small spacing between samples. That is, it has become a *continuous signal*. This is the DTFT, the procedure that changes a discrete aperiodic signal in the time domain into a frequency domain that is a continuous curve. In mathematical terms, a system's frequency response is found by taking the DTFT of its impulse response. Since this cannot be done in a computer, the DFT is used to calculate a *sampling* of the true frequency response. This is the difference between what you do in a computer (the DFT) and what you do with mathematical equations (the DTFT).

Convolution via the Frequency Domain

Suppose that you despise convolution. What are you going to do if given an input signal and impulse response, and need to find the resulting output signal? Figure 9-8 provides an answer: transform the two signals into the frequency domain, multiply them, and then transform the result back into the time domain. This replaces one convolution with two DFTs, a multiplication, and an Inverse DFT. Even though the intermediate steps are very different, the output is *identical* to the standard convolution algorithm.

Does anyone hate convolution enough to go to this trouble? The answer is yes. Convolution is avoided for two reasons. First, convolution is *mathematically* difficult to deal with. For instance, suppose you are given a system's impulse response, and its output signal. How do you calculate what the input signal is? This is called **deconvolution**, and is virtually impossible to understand in the time domain. However, deconvolution can be carried out in the frequency domain as a simple *division*, the inverse operation of multiplication. The frequency domain becomes attractive whenever the complexity of the Fourier Transform is less than the complexity of the convolution. This isn't a matter of which you like better; it is a matter of which you hate less.

The second reason for avoiding convolution is *computation speed*. For example, suppose you design a digital filter with a kernel (impulse response) containing 512 samples. Using a 200 MHz personal computer with floating point numbers, each sample in the output signal requires about one millisecond to calculate, using the standard convolution algorithm. In other words, the throughput of the system is only about 1,000 samples per second. This is 40 times too slow for high-fidelity audio, and 10,000 times too slow for television quality video!

The standard convolution algorithm is slow because of the large number of multiplications and additions that must be calculated. Unfortunately, simply bringing the problem into the frequency domain via the DFT doesn't help at all. Just as many calculations are required to calculate the DFTs, as are required to directly calculate the convolution. A breakthrough was made in the problem in the early 1960s when the *Fast Fourier Transform* (FFT) was developed.

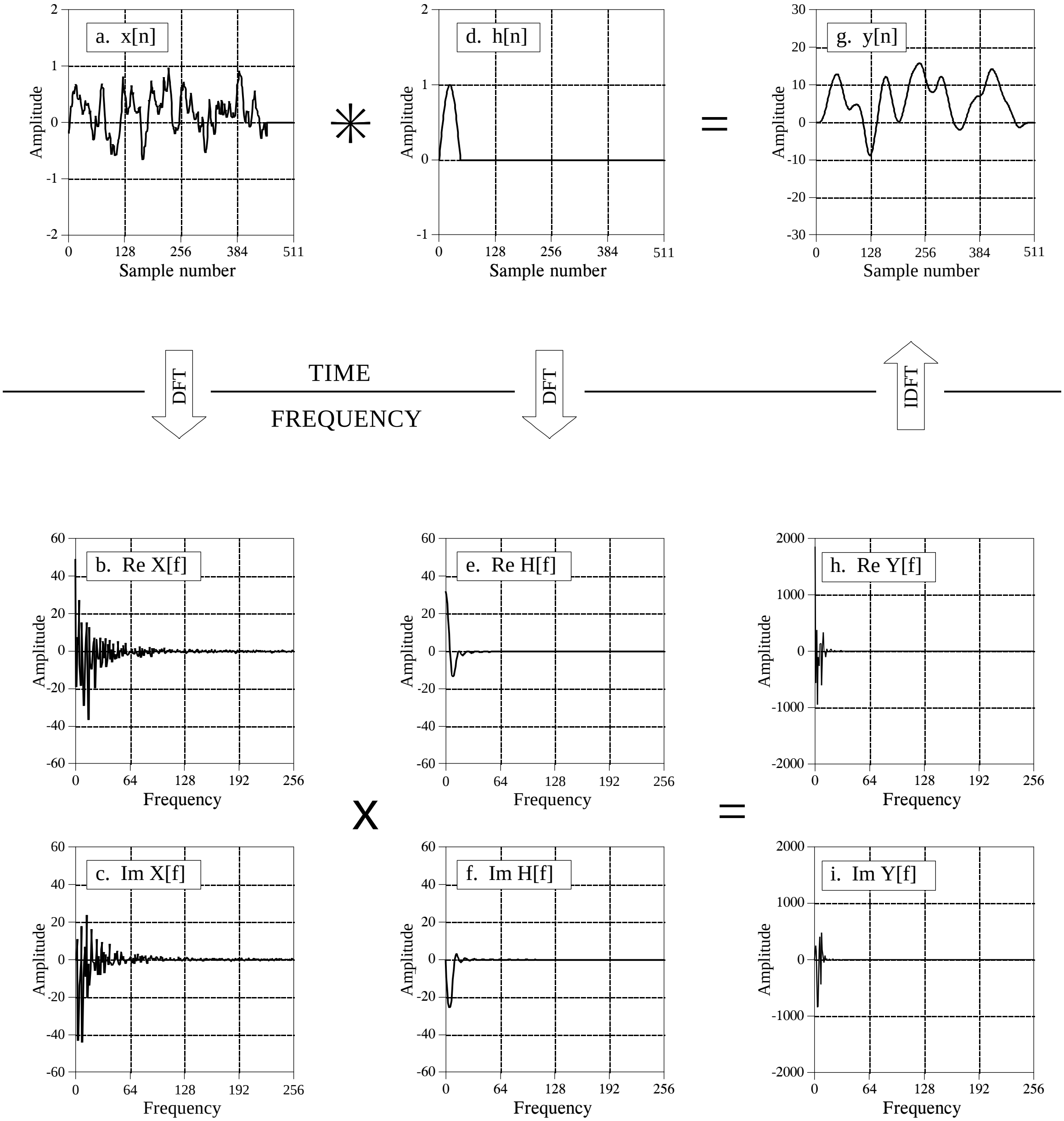

FIGURE 9-8
Frequency domain convolution. In the *time domain*, $x[n]$ is convolved with $h[n]$ resulting in $y[n]$, as is shown in Figs. (a), (d), and (g). This same procedure to be accomplished in the *frequency domain*. The DFT is used to find the frequency spectrum of the input signal, (b) & (c), and the system's frequency response, (e) & (f). Multiplying these two frequency domain signals results in the frequency spectrum of the output signal, (h) & (i). The Inverse DFT is then used to find the output signal, (g).

The FFT is a clever algorithm for rapidly calculating the DFT. Using the FFT, convolution by multiplication in the frequency domain can be hundreds of times faster than conventional convolution. Problems that take hours of calculation time are reduced to only minutes. This is why people get excited about the FFT, and processing signals in the frequency domain. The FFT will be presented in

Chapter 12, and the method of FFT convolution in Chapter 18. For now, focus on how signals are convolved by frequency domain multiplication.

To start, we need to define how to multiply one frequency domain signal by another, i.e., what it means to write: $X[f] \times H[f] = Y[f]$. In polar form, the magnitudes are multiplied: $Mag\,Y[f] = Mag\,X[f] \times Mag\,H[f]$, and the phases are added: $Phase\,Y[f] = Phase\,X[f] + Phase\,H[f]$. To understand this, imagine a cosine wave entering a system with some amplitude and phase. Likewise, the output signal is also a cosine wave with some amplitude and phase. The polar form of the frequency response directly describes how the two amplitudes are related and how the two phases are related.

When frequency domain multiplication is carried out in *rectangular form* there are cross terms between the real and imaginary parts. For example, a sine wave entering the system can produce both cosine and sine waves in the output. To multiply frequency domain signals in rectangular notation:

EQUATION 9-1
Multiplication of frequency domain signals in rectangular form: $Y[f] = X[f] \times H[f]$.

$$Re\,Y[f] = Re\,X[f]\,Re\,H[f] - Im\,X[f]\,Im\,H[f]$$

$$Im\,Y[f] = Im\,X[f]\,Re\,H[f] + Re\,X[f]\,Im\,H[f]$$

Focus on understanding multiplication using *polar notation*, and the idea of cosine waves passing through the system. Then simply accept that these more elaborate equations result when the same operations are carried out in rectangular form. For instance, let's look at the *division* of one frequency domain signal by another. In polar form, the division of frequency domain signals is achieved by the inverse operations we used for multiplication. To calculate: $H[f] = Y[f] / X[f]$, divide the magnitudes and subtract the phases, i.e., $Mag\,H[f] = Mag\,Y[f] / Mag\,X[f]$, $Phase\,H[f] = Phase\,Y[f] - Phase\,X[f]$. In rectangular form this becomes:

EQUATION 9-2
Division of frequency domain signals in rectangular form, where: $H[f] = Y[f] / X[f]$.

$$Re\,H[f] = \frac{Re\,Y[f]\,Re\,X[f] + Im\,Y[f]\,Im\,X[f]}{Re\,X[f]^2 + Im\,X[f]^2}$$

$$Im\,H[f] = \frac{Im\,Y[f]\,Re\,X[f] - Re\,Y[f]\,Im\,X[f]}{Re\,X[f]^2 + Im\,X[f]^2}$$

Now back to frequency domain convolution. You may have noticed that we cheated slightly in Fig. 9-8. Remember, the convolution of an N point signal with an M point impulse response results in an $N+M-1$ point output signal. We cheated by making the last part of the input signal all *zeros* to allow this expansion to occur. Specifically, (a) contains 453 nonzero samples, and (b) contains 60 nonzero samples. This means the convolution of the two, shown in (c), can fit comfortably in the 512 points provided.

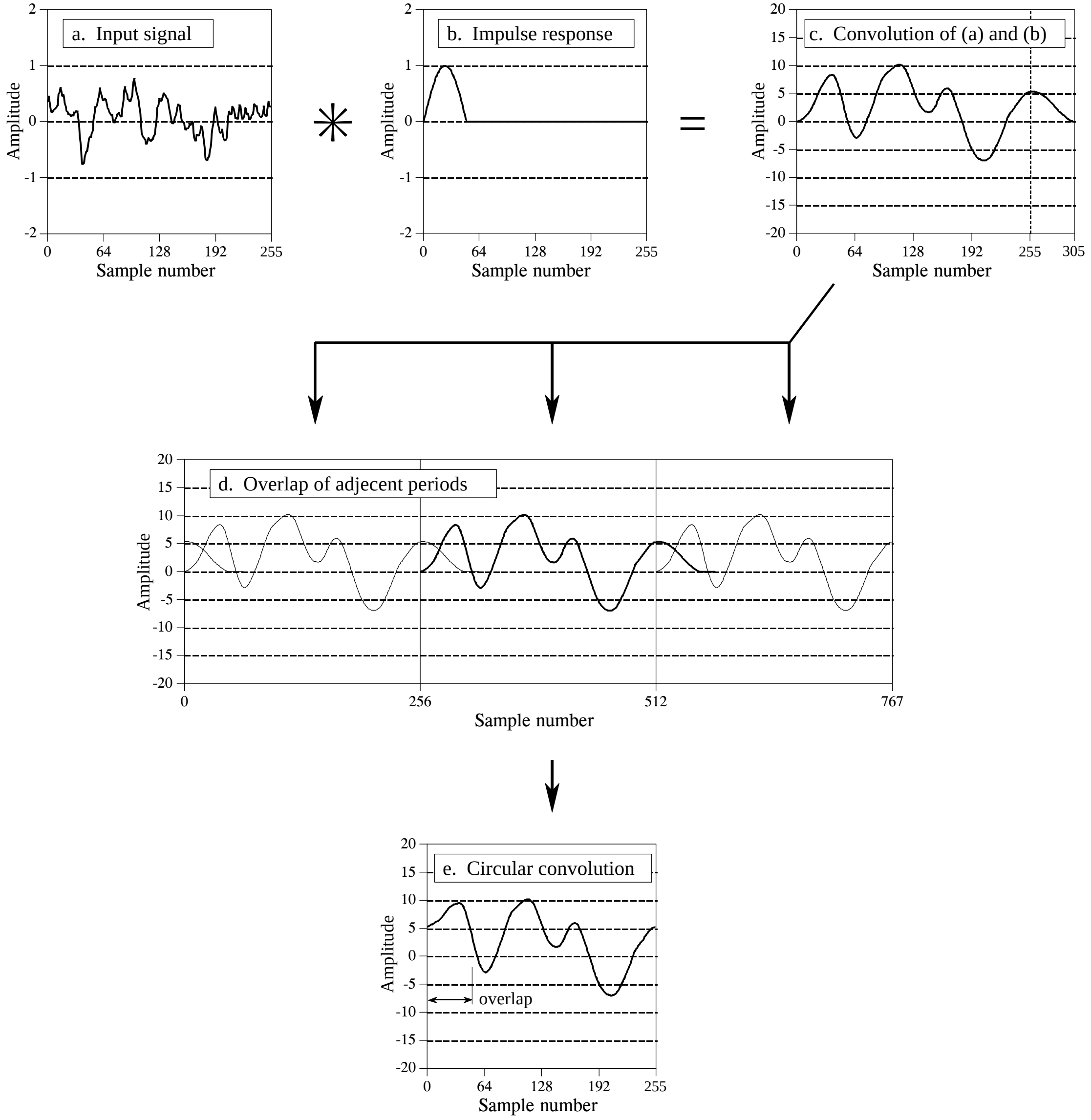

FIGURE 9-9
Circular convolution. A 256 sample signal, (a), convolved with a 51 sample impulse response, (b), results in a 306 sample signal, (c). If this convolution is performed in the frequency domain using 256 point DFTs, the 306 points in the correct convolution cannot fit into the 256 samples provided. As shown in (d), samples 256 through 305 of the output signal are pushed into the next period to the right, where they *add* to the beginning of the next period's signal. Figure (e) is a single period of the resulting signal.

Now consider the more general case in Fig. 9-9. The input signal, (a), is 256 points long, while the impulse response, (b), contains 51 nonzero points. This makes the convolution of the two signals 306 samples long, as shown in (c). The problem is, if we use frequency domain multiplication to perform the convolution, there are only 256 samples *allowed* in the output signal. In other words, 256 point DFTs are used to move (a) and (b) into the frequency domain. After the multiplication, a 256 point Inverse DFT is used to find the output signal. How do you squeeze 306 values of the correct signal into the 256 points provided by the frequency domain algorithm? The answer is, you can't! The 256 points end up being a distorted version of the correct signal. This process is called **circular convolution**. It is important because you want to *avoid* it.

To understand circular convolution, remember that an *N* point DFT views the time domain as being an infinitely long periodic signal, with *N* samples per period. Figure (d) shows three periods of how the DFT views the output signal in this example. Since $N = 256$, each period consists of 256 points: 0-255, 256-511, and 512-767. Frequency domain convolution tries to place the 306 point *correct output signal*, shown in (c), into each of these 256 point periods. This results in 49 of the samples being pushed into the neighboring period to the right, where they overlap with the samples that are legitimately there. These overlapping sections add, resulting in each of the periods appearing as shown in (e), the *circular convolution*.

Once the nature of circular convolution is understood, it is quite easy to avoid. Simply pad each of the signals being convolved with enough zerosto allow the output signal room to handle the $N+M-1$ points in the correct convolution. For example, the signals in (a) and (b) could be padded with zeros to make them 512 points long, allowing the use of 512 point DFTs. After the frequency domain convolution, the output signal would consist of 306 nonzero samples, plus 206 samples with a value of zero. Chapter 18 explains this procedure in detail.

Why is it called *circular* convolution? Look back at Fig. 9-9d and examine the center period, samples 256 to 511. Since all of the periods are the same, the portion of the signal that flows out of this period to the *right*, is the same that flows into this period from the *left*. If you only consider a single period, such as in (e), it *appears* that the right side of the signal is somehow *connected* to the left side. Imagine a snake biting its own tail; sample 255 is located next to sample 0, just as sample 100 is located next to sample 101. When a portion of the signal exits to the right, it magically reappears on the left. In other words, the *N* point time domain behaves as if it were *circular*.

In the last chapter we posed the question: does it really matter if the DFT's time domain is viewed as being *N* points, rather than an infinitely long periodic signal of period *N*? Circular convolution is an example where it *does* matter. If the time domain signal is understood to be *periodic*, the distortion encountered in circular convolution can be simply explained as the signal expanding from one period to the next. In comparison, a rather bizarre conclusion is reached if only *N* points of the time domain are considered. That is, frequency domain convolution acts as if the time domain is somehow wrapping into a circular ring with sample 0 being positioned next to sample *N*-1.