How the FFT works

The FFT is a complicated algorithm, and its details are usually left to those that specialize in such things. This section describes the general operation of the FFT, but skirts a key issue: the use of *complex numbers*. If you have a background in complex mathematics, you can read between the lines to understand the true nature of the algorithm. Don't worry if the details elude you; few scientists and engineers that use the FFT could write the program from scratch.

In complex notation, the time and frequency domains each contain *one signal* made up of *N complex points*. Each of these complex points is composed of two numbers, the real part and the imaginary part. For example, when we talk about complex sample $X[42]$, it refers to the combination of $ReX[42]$ and $ImX[42]$. In other words, each complex variable holds two numbers. When two complex variables are multiplied, the four individual components must be combined to form the two components of the product (such as in Eq. 9-1). The following discussion on *"How the FFT works"* uses this jargon of complex notation. That is, the singular terms: *signal*, *point*, *sample*, and *value*, refer to the *combination* of the real part and the imaginary part.

The FFT operates by decomposing an N point time domain signal into N time domain signals each composed of a single point. The second step is to calculate the N frequency spectra corresponding to these N time domain signals. Lastly, the N spectra are synthesized into a single frequency spectrum.

Figure 12-2 shows an example of the time domain decomposition used in the FFT. In this example, a 16 point signal is decomposed through four

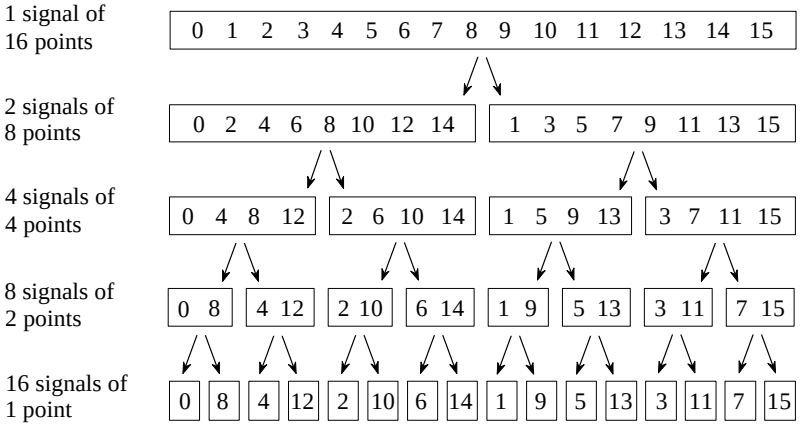

FIGURE 12-2
The FFT decomposition. An N point signal is decomposed into N signals each containing a single point. Each stage uses an *interlace decomposition*, separating the even and odd numbered samples.

Sample numbers in normal order			Sample numbers after bit reversal	
Decimal	*Binary*		*Decimal*	*Binary*
0	0000		0	0000
1	0001		8	1000
2	0010		4	0100
3	0011		12	1100
4	0100		2	0010
5	0101		10	1010
6	0110	⇨	6	0100
7	0111		14	1110
8	1000		1	0001
9	1001		9	1001
10	1010		5	0101
11	1011		13	1101
12	1100		3	0011
13	1101		11	1011
14	1110		7	0111
15	1111		15	1111

FIGURE 12-3
The FFT bit reversal sorting. The FFT time domain decomposition can be implemented by sorting the samples according to bit reversed order.

separate stages. The first stage breaks the 16 point signal into two signals each consisting of 8 points. The second stage decomposes the data into four signals of 4 points. This pattern continues until there are N signals composed of a single point. An **interlaced decomposition** is used each time a signal is broken in two, that is, the signal is separated into its even and odd numbered samples. The best way to understand this is by inspecting Fig. 12-2 until you grasp the pattern. There are $Log_2 N$ stages required in this decomposition, i.e., a 16 point signal (2^4) requires 4 stages, a 512 point signal (2^7) requires 7 stages, a 4096 point signal (2^{12}) requires 12 stages, etc. Remember this value, $Log_2 N$; it will be referenced many times in this chapter.

Now that you understand the structure of the decomposition, it can be greatly simplified. The decomposition is nothing more than a *reordering* of the samples in the signal. Figure 12-3 shows the rearrangement pattern required. On the left, the sample numbers of the original signal are listed along with their binary equivalents. On the right, the rearranged sample numbers are listed, also along with their binary equivalents. The important idea is that the binary numbers are the *reversals* of each other. For example, sample 3 (0011) is exchanged with sample number 12 (1100). Likewise, sample number 14 (1110) is swapped with sample number 7 (0111), and so forth. The FFT time domain decomposition is usually carried out by a **bit reversal sorting** algorithm. This involves rearranging the order of the N time domain samples by counting in binary with the bits flipped left-for-right (such as in the far right column in Fig. 12-3).

The next step in the FFT algorithm is to find the frequency spectra of the 1 point time domain signals. Nothing could be easier; the frequency spectrum of a 1 point signal is equal to *itself*. This means that *nothing* is required to do this step. Although there is no work involved, don't forget that each of the 1 point signals is now a frequency spectrum, and not a time domain signal.

The last step in the FFT is to combine the *N* frequency spectra in the exact reverse order that the time domain decomposition took place. This is where the algorithm gets messy. Unfortunately, the bit reversal shortcut is not applicable, and we must go back one stage at a time. In the first stage, 16 frequency spectra (1 point each) are synthesized into 8 frequency spectra (2 points each). In the second stage, the 8 frequency spectra (2 points each) are synthesized into 4 frequency spectra (4 points each), and so on. The last stage results in the output of the FFT, a 16 point frequency spectrum.

Figure 12-4 shows how two frequency spectra, each composed of 4 points, are combined into a single frequency spectrum of 8 points. This synthesis must *undo* the interlaced decomposition done in the time domain. In other words, the frequency domain operation must correspond to the time domain procedure of *combining* two 4 point signals by interlacing. Consider two time domain signals, *abcd* and *efgh*. An 8 point time domain signal can be formed by two steps: dilute each 4 point signal with zeros to make it an

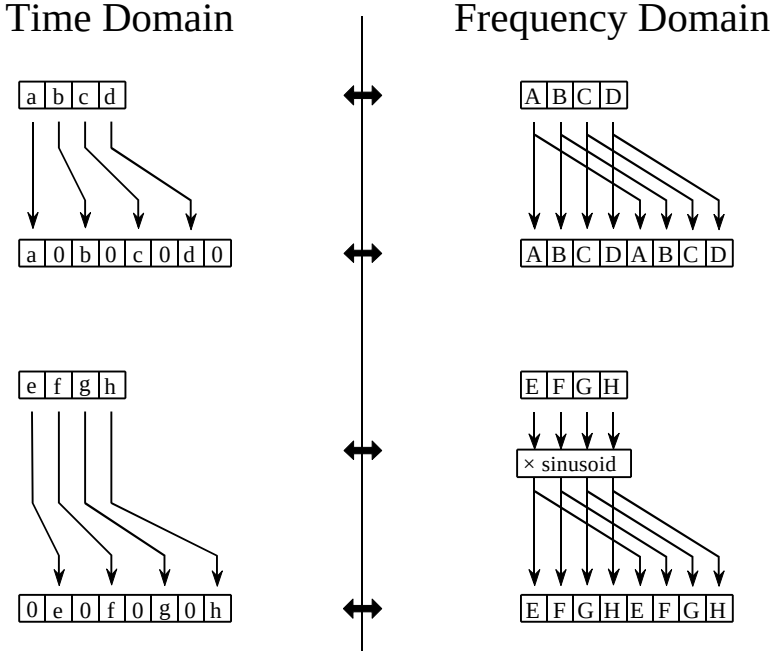

FIGURE 12-4
The FFT synthesis. When a time domain signal is diluted with zeros, the frequency domain is duplicated. If the time domain signal is also shifted by one sample during the dilution, the spectrum will additionally be multiplied by a sinusoid.

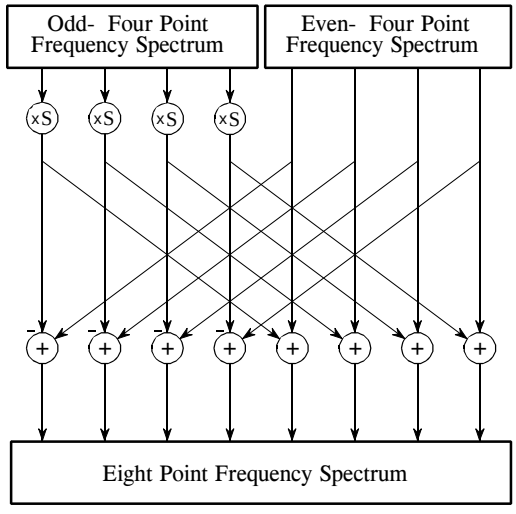

FIGURE 12-5
FFT synthesis flow diagram. This shows the method of combining two 4 point frequency spectra into a single 8 point frequency spectrum. The ×S operation means that the signal is multiplied by a sinusoid with an appropriately selected frequency.

8 point signal, and then add the signals together. That is, *abcd* becomes *a0b0c0d0*, and *efgh* becomes *0e0f0g0h*. Adding these two 8 point signals produces *aebfcgdh*. As shown in Fig. 12-4, diluting the time domain with zeros corresponds to a *duplication* of the frequency spectrum. Therefore, the frequency spectra are combined in the FFT by duplicating them, and then adding the duplicated spectra together.

In order to match up when added, the two time domain signals are diluted with zeros in a slightly different way. In one signal, the *odd points* are zero, while in the other signal, the *even points* are zero. In other words, one of the time domain signals (*0e0f0g0h* in Fig. 12-4) is shifted to the right by one sample. This time domain shift corresponds to multiplying the spectrum by a *sinusoid*. To see this, recall that a shift in the time domain is equivalent to convolving the signal with a shifted delta function. This multiplies the signal's spectrum with the spectrum of the shifted delta function. The spectrum of a shifted delta function is a sinusoid (see Fig 11-2).

Figure 12-5 shows a flow diagram for combining two 4 point spectra into a single 8 point spectrum. To reduce the situation even more, notice that Fig. 12-5 is formed from the basic pattern in Fig 12-6 repeated over and over.

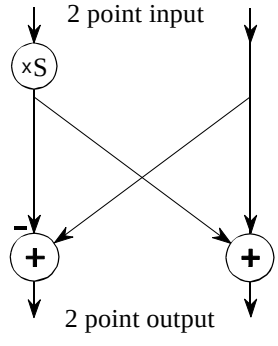

FIGURE 12-6
The FFT butterfly. This is the basic calculation element in the FFT, taking two complex points and converting them into two other complex points.

This simple flow diagram is called a **butterfly** due to its winged appearance. The butterfly is the basic computational element of the FFT, transforming two complex points into two other complex points.

Figure 12-7 shows the structure of the entire FFT. The time domain decomposition is accomplished with a bit reversal sorting algorithm. Transforming the decomposed data into the frequency domain involves *nothing* and therefore does not appear in the figure.

The frequency domain synthesis requires three loops. The outer loop runs through the Log_2N stages (i.e., each level in Fig. 12-2, starting from the bottom and moving to the top). The middle loop moves through each of the individual frequency spectra in the stage being worked on (i.e., each of the boxes on any one level in Fig. 12-2). The innermost loop uses the butterfly to calculate the points in each frequency spectra (i.e., looping through the samples inside any one box in Fig. 12-2). The overhead boxes in Fig. 12-7 determine the beginning and ending indexes for the loops, as well as calculating the sinusoids needed in the butterflies. Now we come to the heart of this chapter, the actual FFT programs.

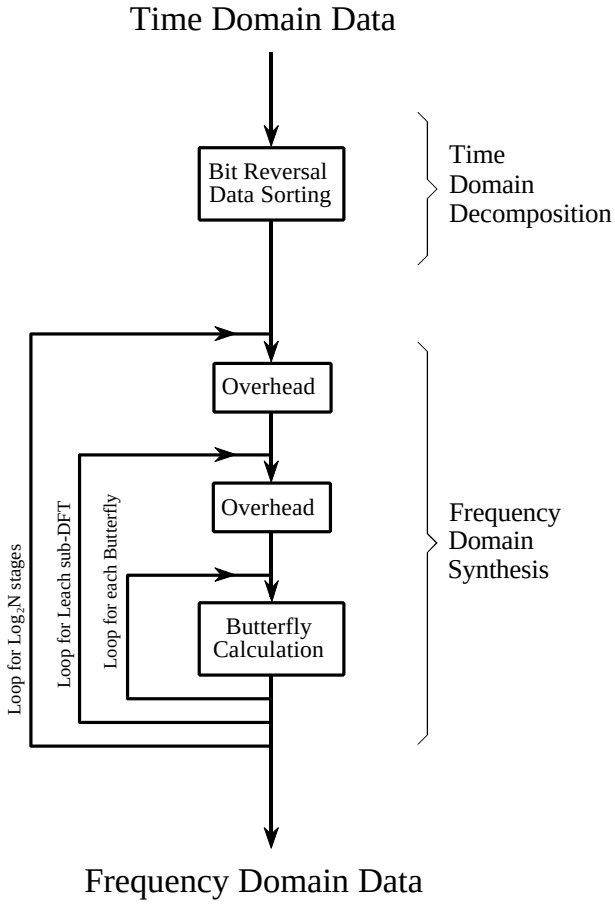

FIGURE 12-7
Flow diagram of the FFT. This is based on three steps: (1) decompose an N point time domain signal into N signals each containing a single point, (2) find the spectrum of each of the N point signals (nothing required), and (3) synthesize the N frequency spectra into a single frequency spectrum.