SECTION 5

FAST FOURIER TRANSFORMS

- The Discrete Fourier Transform
- The Fast Fourier Transform
- FFT Hardware Implementation and Benchmarks
- DSP Requirements for Real Time FFT Applications
- Spectral Leakage and Windowing

SECTION 5
FAST FOURIER TRANSFORMS
Walt Kester

THE DISCRETE FOURIER TRANSFORM

In 1807 the French mathematician and physicist Jean Baptiste Joseph Fourier presented a paper to the *Institut de France* on the use of sinusoids to represent temperature distributions. The paper made the controversial claim that *any continuous periodic signal could be represented by the sum of properly chosen sinusoidal waves*. Among the publication review committee were two famous mathematicians: Joseph Louis Lagrange, and Pierre Simon de Laplace. Lagrange objected strongly to publication on the basis that Fourier's approach would not work with signals having discontinuous slopes, such as square waves. Fourier's work was rejected, primarily because of Lagrange's objection, and was not published until the death of Lagrange, some 15 years later. In the meantime, Fourier's time was occupied with political activities, expeditions to Egypt with Napoleon, and trying to avoid the guillotine after the French Revolution! (This bit of history extracted from Reference 1, p.141).

It turns out that both Fourier and Lagrange were at least partially correct. Lagrange was correct that a summation of sinusoids cannot *exactly* form a signal with a corner. However, you can get *very* close if enough sinusoids are used. (This is described by the *Gibbs effect,* and is well understood by scientists, engineers, and mathematicians today).

Fourier analysis forms the basis for much of digital signal processing. Simply stated, the Fourier transform (there are actually several members of this family) allows a time domain signal to be converted into its equivalent representation in the frequency domain. Conversely, if the frequency response of a signal is known, the inverse Fourier transform allows the corresponding time domain signal to be determined.

In addition to frequency analysis, these transforms are useful in filter design, since the frequency response of a filter can be obtained by taking the Fourier transform of its impulse response. Conversely, if the frequency response is specified, then the required impulse response can be obtained by taking the inverse Fourier transform of the frequency response. Digital filters can be constructed based on their impulse response, because the coefficients of an FIR filter and its impulse response are identical.

The Fourier transform family (*Fourier Transform*, *Fourier Series, Discrete Time Fourier Series*, and *Discrete Fourier Transform*) is shown in Figure 5.2. These accepted definitions have evolved (not necessarily logically) over the years and depend upon whether the signal is *continuous–aperiodic*, *continuous–periodic*, *sampled–aperiodic, or sampled–periodic*. In this context, the term *sampled* is the same as *discrete* (i.e., a *discrete* number of time samples).

APPLICATIONS OF THE DISCRETE FOURIER TRANSFORM (DFT)

Sampled Time Domain → Discrete Fourier Transform (DFT) → **Sampled Frequency Domain**

Sampled Frequency Domain → Inverse DFT (IDFT) → **Sampled Time Domain**

- **Digital Spectral Analysis**
  - **Spectrum Analyzers**
  - **Speech Processing**
  - **Imaging**
  - **Pattern Recognition**
- **Filter Design**
  - **Calculating Impulse Response from Frequency Response**
  - **Calculating Frequency Response from Impulse Response**
- **The Fast Fourier Transform (FFT) is Simply an Algorithm for Efficiently Calculating the DFT**

Figure 5.1

FOURIER TRANSFORM FAMILY AS A FUNCTION OF TIME DOMAIN SIGNAL TYPE

FOURIER TRANSFORM:
Signal is Continuous and Aperiodic

FOURIER SERIES:
Signal is Continuous and Periodic

DISCRETE TIME FOURIER SERIES:
Signal is Sampled and Aperiodic

DISCRETE FOURIER TRANSFORM:
(Discrete Fourier Series)
Signal is Sampled and Periodic

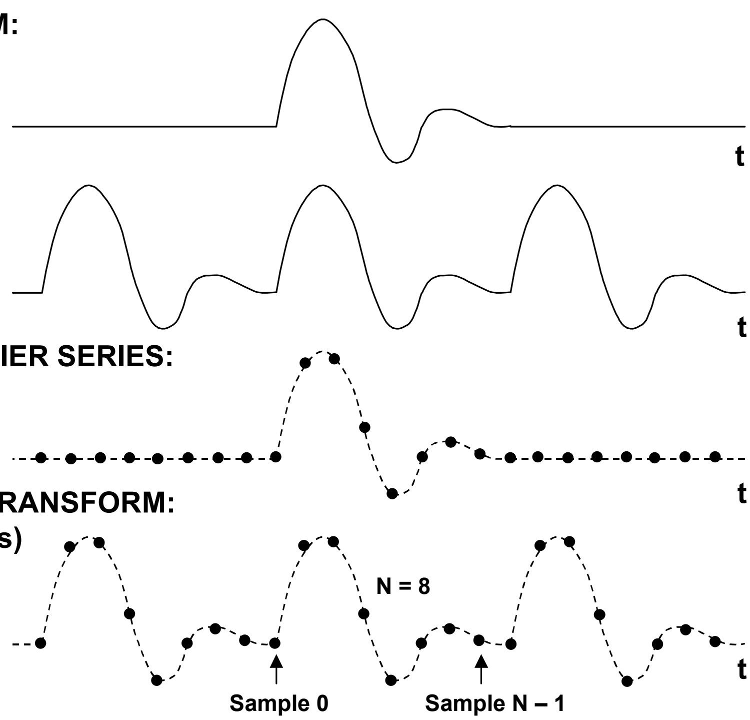

Figure 5.2

The only member of this family which is relevant to digital signal processing is the *Discrete Fourier Transform (DFT)* which operates on a *sampled* time domain signal which is *periodic*. The signal must be periodic in order to be decomposed into the summation of sinusoids. However, only a finite number of samples (N) are available for inputting into the DFT. This dilemma is overcome by placing an infinite number of groups of the same N samples "end-to-end," thereby forcing mathematical (but not real-world) periodicity as shown in Figure 5.2.

The fundamental analysis equation for obtaining the N-point DFT is as follows:

$$X(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)e^{-j2\pi nk/N} = \frac{1}{N}\sum_{n=0}^{N-1} x(n)\left[\cos(2\pi nk/N) - j\sin(2\pi nk/N)\right]$$

At this point, some terminology clarifications are in order regarding the above equation (also see Figure 5.3). X(k) (capital letter X) represents the DFT frequency output at the kth spectral point, where k ranges from 0 to N–1. The quantity N represents the number of sample points in the DFT data frame.

Note that "N" should not be confused with ADC or DAC resolution, which is also given by the quantity N in other places in this book.

The quantity x(n) (lower case letter x) represents the nth time sample, where n also ranges from 0 to N – 1. In the general equation, x(n) can be real or complex.

Notice that the cosine and sine terms in the equation can be expressed in either polar or rectangular coordinates using Euler's equation:

$$e^{j\theta} = \cos\theta + j\sin\theta$$

The DFT output spectrum, X(k), is the correlation between the input time samples and N cosine and N sine waves. The concept is best illustrated in Figure 5.4. In this figure, the real part of the first four output frequency points is calculated, therefore, only the cosine waves are shown. A similar procedure is used with sine waves in order to calculate the imaginary part of the output spectrum.

The first point, X(0), is simply the sum of the input time samples, because cos(0) = 1. The scaling factor, 1/N, is not shown, but must be present in the final result. Note that X(0) is the average value of the time samples, or simply the DC offset. The second point, ReX(1), is obtained by multiplying each time sample by each corresponding point on a cosine wave which makes one complete cycle in the interval N and summing the results. The third point, ReX(2), is obtained by multiplying each time sample by each corresponding point of a cosine wave which has two complete cycles in the interval N and then summing the results. Similarly, the fourth point, ReX(3), is obtained by multiplying each time sample by the corresponding point of a cosine wave which has three complete cycles in the interval N and summing the results. This process continues until all N outputs have been computed. A similar procedure is followed using sine waves in order to calculate the imaginary part of the frequency spectrum. The cosine and sine waves are referred to as *basis functions*.

THE DISCRETE FOURIER TRANSFORM (DFT)

- A Periodic Signal Can be Decomposed into the Sum of Properly Chosen Cosine and Sine Waves (Jean Baptiste Joseph Fourier, 1807)

- The DFT Operates on a Finite Number (N) of Digitized Time Samples, x(n). When These Samples are Repeated and Placed "End-to-End", they Appear Periodic to the Transform.

- The Complex DFT Output Spectrum X(k) is the Result of Correlating the Input Samples with sine and cosine Basis Functions:

$$X(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)\, e^{\frac{-j2\pi nk}{N}} = \frac{1}{N}\sum_{n=0}^{N-1} x(n)\left[\cos\frac{2\pi nk}{N} - j\sin\frac{2\pi nk}{N}\right]$$

$$0 \le k \le N-1$$

Figure 5.3

CORRELATION OF TIME SAMPLES WITH BASIS FUNCTIONS USING THE DFT FOR N = 8

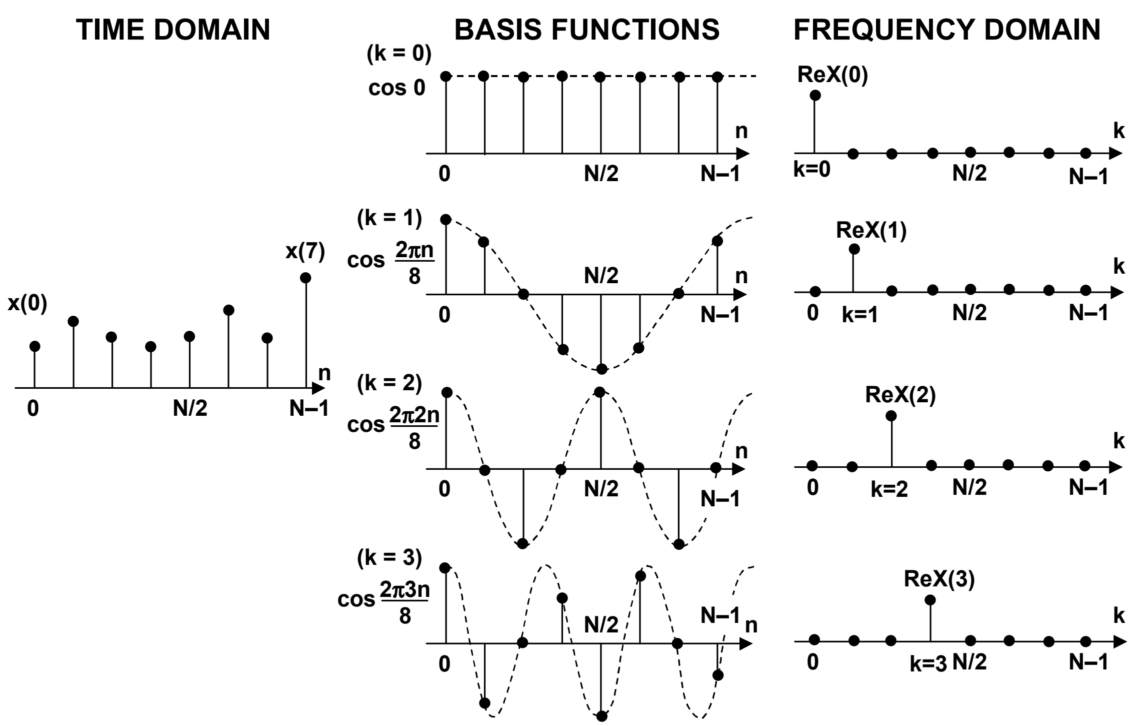

Figure 5.4

Assume that the input signal is a cosine wave having a period of N, i.e., it makes one complete cycle during the data window. Also assume its amplitude and phase is identical to the first cosine wave of the basis functions, cos(2πn/8). The output ReX(1) contains a single point, and all the other ReX(k) outputs are zero. Assume that the input cosine wave is now shifted to the right by 90°. The correlation between it and the corresponding basis function is zero. However, there is an additional correlation required with the basis function sin(2πn/8) to yield ImX(1). This shows why both real and imaginary parts of the frequency spectrum need to be calculated in order to determine both the amplitude and phase of the frequency spectrum.

Notice that the correlation of a sine/cosine wave of any frequency other than that of the basis function produces a zero value for both ReX(1) and ImX(1).

A similar procedure is followed when using the *inverse* DFT (IDFT) to reconstruct the time domain samples, x(n), from the frequency domain samples X(k). The synthesis equation is given by:

$$x(n) = \sum_{k=0}^{N-1} X(k)e^{j2\pi nk/N} = \sum_{k=0}^{N-1} X(k)\left[\cos(2\pi nk/N) + j\sin(2\pi nk/N)\right]$$

There are two basic types of DFTs: *real*, and *complex*. The equations shown in Figure 5.5 are for the complex DFT, where the input and output are both complex numbers. Since time domain input samples are real and have no imaginary part, the imaginary part of the input is always set to zero. The output of the DFT, X(k), contains a real and imaginary component which can be converted into ampltude and phase.

The *real* DFT, although somewhat simpler, is basically a simplification of the *complex* DFT. Most FFT routines are written using the complex DFT format, therefore understanding the complex DFT and how it relates to the real DFT is important. For instance, if you know the real DFT frequency outputs and want to use a complex inverse DFT to calculate the time samples, you need to know how to place the real DFT oututs points into the complex DFT format before taking the complex inverse DFT.

Figure 5.6 shows the input and output of a real and a complex FFT. Notice that the output of the real DFT yields real and imaginary X(k) values, where k ranges from only 0 to N/2. Note that the imaginary points ImX(0) and ImX(N/2) are always zero because sin(0) and sin(nπ) are both always zero.

The frequency domain output X(N/2) corresponds to the frequency output at one-half the sampling frequency, f_S. The width of each frequency bin is equal to f_S/N.

THE COMPLEX DFT

Frequency Domain ← ← DFT ← ← Time Domain

$$X(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)\, e^{\frac{-j2\pi nk}{N}} = \frac{1}{N}\sum_{n=0}^{N-1} x(n)\left[\cos\frac{2\pi nk}{N} - j\sin\frac{2\pi nk}{N}\right]$$

$$\boxed{W_N = e^{\frac{-j2\pi}{N}}} \qquad = \frac{1}{N}\sum_{n=0}^{N-1} x(n)\, W_N^{nk}\;, \quad 0 \le k \le N-1$$

Time Domain ← ← INVERSE DFT ← ← Frequency Domain

$$x(n) = \sum_{k=0}^{N-1} X(k)\, e^{\frac{j2\pi nk}{N}} = \sum_{k=0}^{N-1} X(k)\left[\cos\frac{2\pi nk}{N} + j\sin\frac{2\pi nk}{N}\right]$$

$$= \sum_{k=0}^{N-1} X(k)\, W_N^{-nk}\;, \quad 0 \le n \le N-1$$

Figure 5.5

DFT INPUT/OUTPUT SPECTRUM

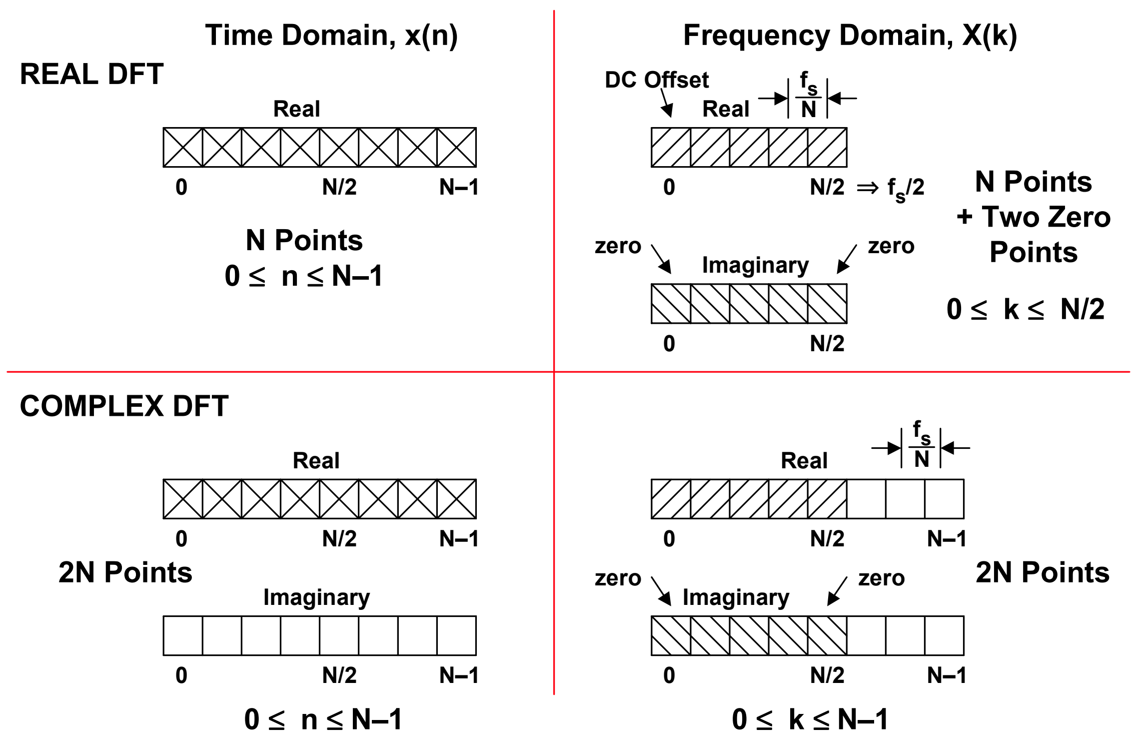

Figure 5.6

The complex DFT has real and imaginary values both at its input and output. In practice, the imaginary parts of the time domain samples are set to zero. If you are given the output spectrum for a complex DFT, it is useful to know how to relate them to the real DFT output and vice versa. The crosshatched areas in the diagram correspond to points which are common to both the real and complex DFT.

Figure 5.7 shows the relationship between the real and complex DFT in more detail. The real DFT output points are from 0 to N/2, with ImX(0) and ImX(N/2) always zero. The points between N/2 and N – 1 contain the negative frequencies in the complex DFT. Note that ReX(N/2 + 1) has the same value as Re(N/2 – 1). Similarly, ReX(N/2 + 2) has the same value as ReX(N/2 – 2), etc. Also, note that ImX(N/2 + 1) is the negative of ImX(N/2 – 1), and ImX(N/2 + 2) is the negative of ImX(N/2 – 2), etc. In other words, ReX(k) has *even symmetry* about N/2 and ImX(k) *has odd symmetry* about N/2. In this way, the negative frequency components for the complex FFT can be generated if you are only given the real DFT components.

The equations for the complex and the real DFT are summarized in Figure 5.8. Note that the equations for the complex DFT work nearly the same whether taking the DFT, X(k) or the IDFT, x(n). The real DFT does not use complex numbers, and the equations for X(k) and x(n) are significantly different. Also, before using the x(n) equation, ReX(0) and ReX(N/2) must be divided by two. These details are explained in Chapter 31 of Reference 1, and the reader should study this chapter before attemping to use these equations.

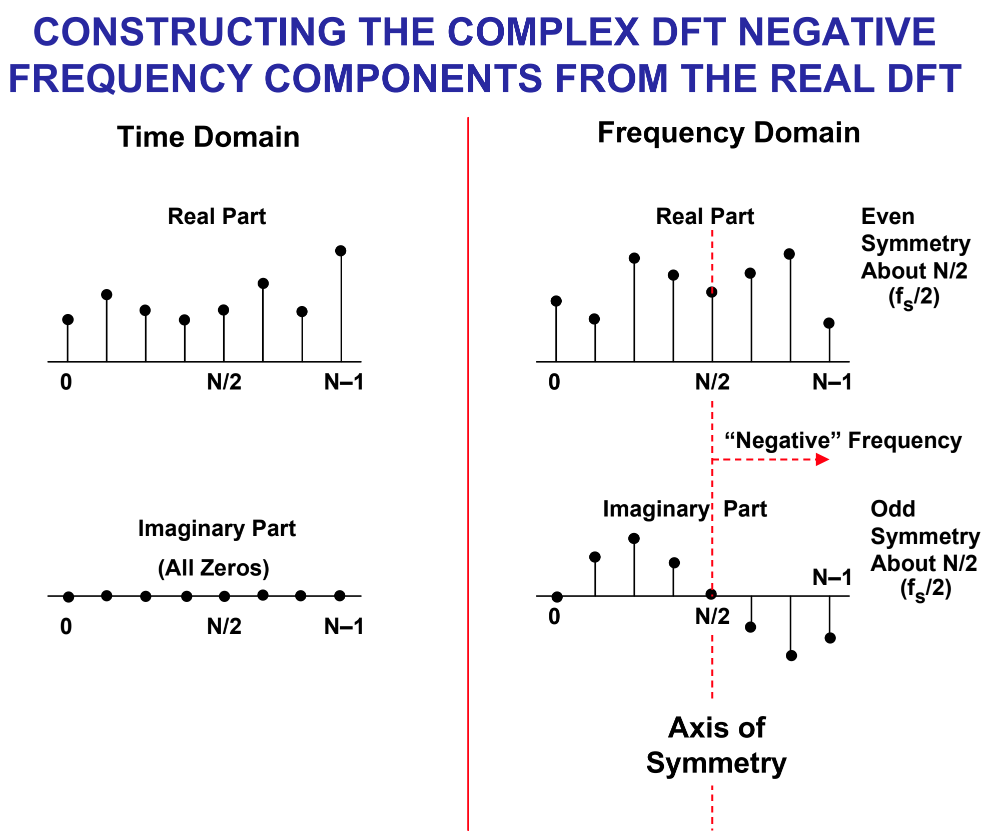

Figure 5.7

The DFT output spectrum can be represented in either polar form (magnitude and phase) or rectangular form (real and imaginary) as shown in Figure 5.9. The conversion between the two forms is straightforward.

COMPLEX AND REAL DFT EQUATIONS

COMPLEX TRANSFORM

$$X(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)\, e^{\frac{-j2\pi nk}{N}}$$

$$x(n) = \sum_{k=0}^{N-1} X(k)\, e^{\frac{j2\pi nk}{N}}$$

REAL TRANSFORM

$$ReX(k) = \frac{2}{N}\sum_{n=0}^{N-1} x(n)\cos(2\pi nk/N)$$

$$ImX(k) = \frac{-2}{N}\sum_{n=0}^{N-1} x(n)\sin(2\pi nk/N)$$

$$x(n) = \sum_{k=0}^{N/2}\left[ReX(k)\cos(2\pi nk/N) - ImX(k)\cos(2\pi nk/N)\right]$$

Time Domain: x(n) is complex, discrete, and periodic. n runs from 0 to N– 1

Frequency Domain: X(k) is complex, discrete, and periodic. k runs from 0 to N–1
k = 0 to N/2 are positive frequencies.
k = N/2 to N–1 are negative frequencies

Time Domain: x(n) is real, discrete, and periodic. n runs from 0 to N – 1

Frequency domain:
ReX(k) is real, discrete, and periodic.
ImX(k) is real, discrete, and periodic.
k runs from 0 to N/2

Before using x(n) equation, ReX(0) and ReX(N/2) must be divided by two.

Figure 5.8

CONVERTING REAL AND IMAGINARY DFT OUTPUTS INTO MAGNITUDE AND PHASE

- $X(k) = ReX(k) + j\, ImX(k)$
- $MAG\,[X(k)] = \sqrt{ReX(k)^2 + ImX(k)^2}$
- $\varphi\,[X(k)] = \tan^{-1}\dfrac{ImX(k)}{ReX(k)}$

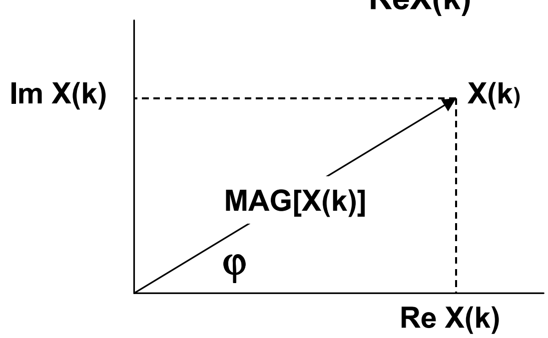

Figure 5.9

THE FAST FOURIER TRANSFORM

In order to understand the development of the FFT, consider first the 8-point DFT expansion shown in Figure 5.10. In order to simplify the diagram, note that the quantity W_N is defined as:

$$W_N = e^{-j2\pi/N}.$$

This leads to the definition of the *twiddle factors* as:

$$W_N{}^{nk} = e^{-j2\pi nk/N}.$$

The twiddle factors are simply the sine and cosine basis functions written in polar form. Note that the 8-point DFT shown in the diagram requires 64 complex multiplications. In general, an N-point DFT requires N^2 complex multiplications. The number of multiplications required is significant because the multiplication function requires a relatively large amount of DSP processing time. In fact, the total time required to compute the DFT is directly proportional to the number of multiplications plus the required amount of overhead.

THE 8-POINT DFT (N = 8)

$$X(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)\, e^{\frac{-j2\pi nk}{N}} = \frac{1}{N}\sum_{n=0}^{N-1} x(n)\, W_N{}^{nk}$$

$$\boxed{W_N = e^{\frac{-j2\pi}{N}}}$$

X(0) =	$x(0)W_8{}^0$	$+ x(1)W_8{}^0$	$+ x(2)W_8{}^0$	$+ x(3)W_8{}^0$	$+ x(4)W_8{}^0$	$+ x(5)W_8{}^0$	$+ x(6)W_8{}^0$	$+ x(7)W_8{}^0$
X(1) =	$x(0)W_8{}^0$	$+ x(1)W_8{}^1$	$+ x(2)W_8{}^2$	$+ x(3)W_8{}^3$	$+ x(4)W_8{}^4$	$+ x(5)W_8{}^5$	$+ x(6)W_8{}^6$	$+ x(7)W_8{}^7$
X(2) =	$x(0)W_8{}^0$	$+ x(1)W_8{}^2$	$+ x(2)W_8{}^4$	$+ x(3)W_8{}^6$	$+ x(4)W_8{}^8$	$+ x(5)W_8{}^{10}$	$+ x(6)W_8{}^{12}$	$+ x(7)W_8{}^{14}$
X(3) =	$x(0)W_8{}^0$	$+ x(1)W_8{}^3$	$+ x(2)W_8{}^6$	$+ x(3)W_8{}^9$	$+ x(4)W_8{}^{12}$	$+ x(5)W_8{}^{15}$	$+ x(6)W_8{}^{18}$	$+ x(7)W_8{}^{21}$
X(4) =	$x(0)W_8{}^0$	$+ x(1)W_8{}^4$	$+ x(2)W_8{}^8$	$+ x(3)W_8{}^{12}$	$+ x(4)W_8{}^{16}$	$+ x(5)W_8{}^{20}$	$+ x(6)W_8{}^{24}$	$+ x(7)W_8{}^{28}$
X(5) =	$x(0)W_8{}^0$	$+ x(1)W_8{}^5$	$+ x(2)W_8{}^{10}$	$+ x(3)W_8{}^{15}$	$+ x(4)W_8{}^{20}$	$+ x(5)W_8{}^{25}$	$+ x(6)W_8{}^{30}$	$+ x(7)W_8{}^{35}$
X(6) =	$x(0)W_8{}^0$	$+ x(1)W_8{}^6$	$+ x(2)W_8{}^{12}$	$+ x(3)W_8{}^{18}$	$+ x(4)W_8{}^{24}$	$+ x(5)W_8{}^{30}$	$+ x(6)W_8{}^{36}$	$+ x(7)W_8{}^{42}$
X(7) =	$x(0)W_8{}^0$	$+ x(1)W_8{}^7$	$+ x(2)W_8{}^{14}$	$+ x(3)W_8{}^{21}$	$+ x(4)W_8{}^{28}$	$+ x(5)W_8{}^{35}$	$+ x(6)W_8{}^{42}$	$+ x(7)W_8{}^{49}$

N^2 Complex Multiplications

$\frac{1}{N}$ Scaling Factor Omitted

Figure 5.10

The FFT is simply an algorithm to speed up the DFT calculation by reducing the number of multiplications and additions required. It was popularized by J. W. Cooley and J. W. Tukey in the 1960s and was actually a rediscovery of an idea of Runge (1903) and Danielson and Lanczos (1942), first occurring prior to the availability of computers and calculators – when numerical calculation could take many man hours. In addition, the German mathematician Karl Friedrich Gauss (1777 – 1855) had used the method more than a century earlier.

In order to understand the basic concepts of the FFT and its derivation, note that the DFT expansion shown in Figure 5.10 can be greatly simplified by taking advantage of the symmetry and periodicity of the twiddle factors as shown in Figure 5.11. If the equations are rearranged and factored, the result is the Fast Fourier Transform (FFT) which requires only (N/2) $\log_2$(N) complex multiplications. The computational efficiency of the FFT versus the DFT becomes highly significant when the FFT point size increases to several thousand as shown in Figure 5.12. However, notice that the FFT computes *all* the output frequency components (either all or none!). If only a few spectral points need to be calculated, the DFT may actually be more efficient. Calculation of a single spectral output using the DFT requires only N complex multiplications.

APPLYING THE PROPERTIES OF SYMMETRY AND PERIODICITY TO W_N^r FOR N = 8

Symmetry: $W_N^{r+N/2} = -W_N^r$, Periodicity: $W_N^{r+N} = W_N^r$

N = 8

$$W_8^4 = W_8^{0+4} = -W_8^0 = -1$$
$$W_8^5 = W_8^{1+4} = -W_8^1$$
$$W_8^6 = W_8^{2+4} = -W_8^2$$
$$W_8^7 = W_8^{3+4} = -W_8^3$$
$$W_8^8 = W_8^{0+8} = +W_8^0 = +1$$
$$W_8^9 = W_8^{1+8} = +W_8^1$$
$$W_8^{10} = W_8^{2+8} = +W_8^2$$
$$W_8^{11} = W_8^{3+8} = +W_8^3$$
$$\vdots \qquad \vdots \qquad \vdots$$

Figure 5.11

THE FAST FOURIER TRANSFORM (FFT) VS. THE DISCRETE FOURIER TRANSFORM (DFT)

- **The FFT is Simply an Algorithm for Efficiently Calculating the DFT**
- **Computational Efficiency of an N-Point FFT:**
  - **DFT:** N^2 **Complex Multiplications**
  - **FFT:** $(N/2)\log_2(N)$ **Complex Multiplications**

N	DFT Multiplications	FFT Multiplications	FFT Efficiency
256	65,536	1,024	64 : 1
512	262,144	2,304	114 : 1
1,024	1,048,576	5,120	205 : 1
2,048	4,194,304	11,264	372 : 1
4,096	16,777,216	24,576	683 : 1

Figure 5.12

The radix-2 FFT algorithm breaks the entire DFT calculation down into a number of 2-point DFTs. Each 2-point DFT consists of a multiply-and-accumulate operation called a *butterfly*, as shown in Figure 5.13. Two representations of the butterfly are shown in the diagram: the top diagram is the actual functional representation of the butterfly showing the digital multipliers and adders. In the simplified bottom diagram, the multiplications are indicated by placing the multiplier over an arrow, and addition is indicated whenever two arrows converge at a dot.

The 8-point decimation-in-time (DIT) FFT algorithm computes the final output in three stages as shown in Figure 5.14. The eight input time samples are first divided (or *decimated*) into four groups of 2-point DFTs. The four 2-point DFTs are then combined into two 4-point DFTs. The two 4-point DFTs are then combined to produce the final output X(k). The detailed process is shown in Figure 5.15, where all the multiplications and additions are shown. Note that the basic two-point DFT butterfly operation forms the basis for all computation. The computation is done in three stages. After the first stage computation is complete, there is no need to store any previous results. The first stage outputs can be stored in the same registers which originally held the time samples x(n). Similarly, when the second stage computation is completed, the results of the first stage computation can be deleted. In this way, *in-place* computation proceeds to the final stage. Note that in order for the algorithm to work properly, the order of the input time samples, x(n), must be properly re-ordered using a *bit reversal* algorithm.

THE BASIC BUTTERFLY COMPUTATION IN THE DECIMATION-IN-TIME FFT ALGORITHM

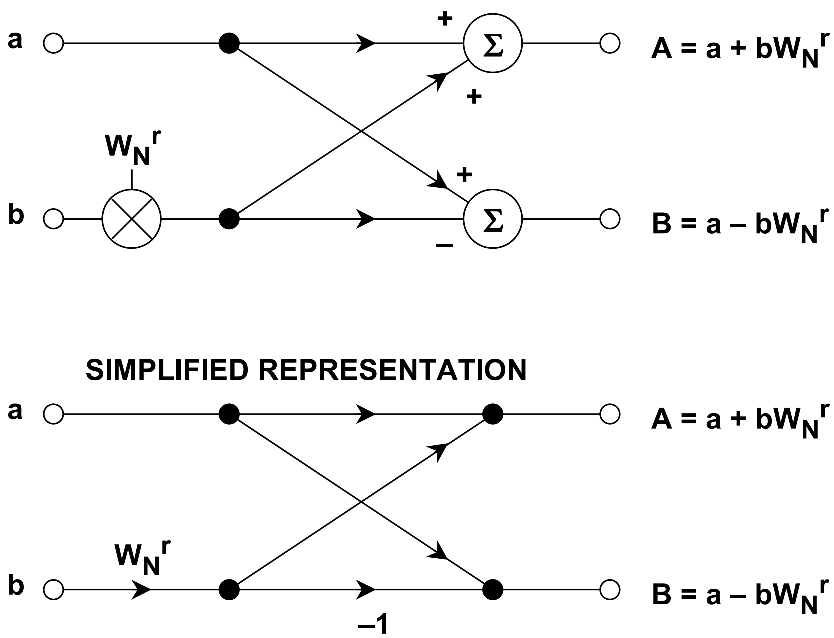

Figure 5.13

COMPUTATION OF AN 8-POINT DFT IN THREE STAGES USING DECIMATION-IN-TIME

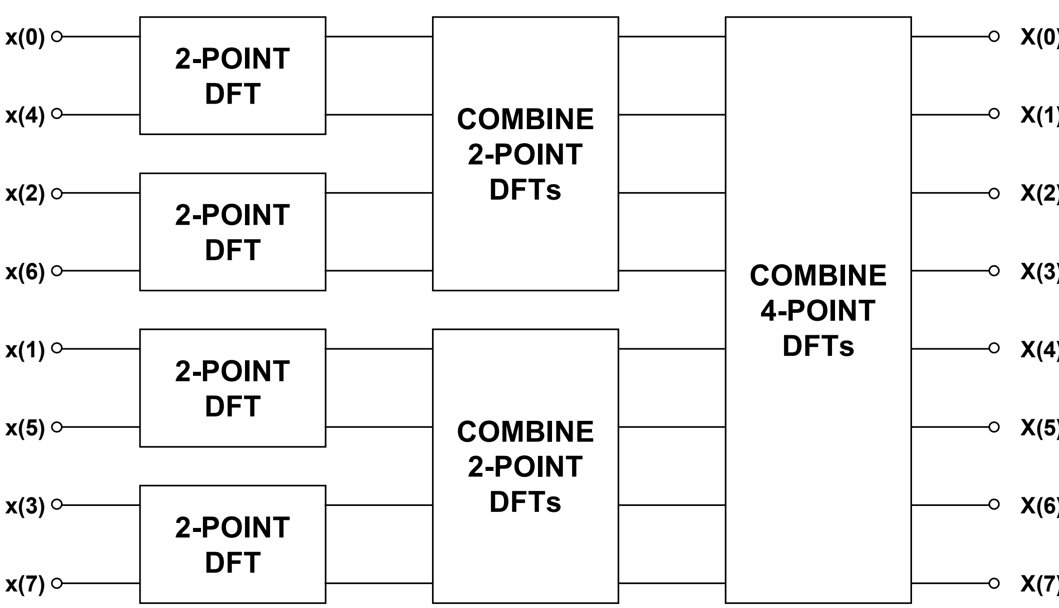

Figure 5.14

EIGHT-POINT DECIMATION-IN-TIME FFT ALGORITHM

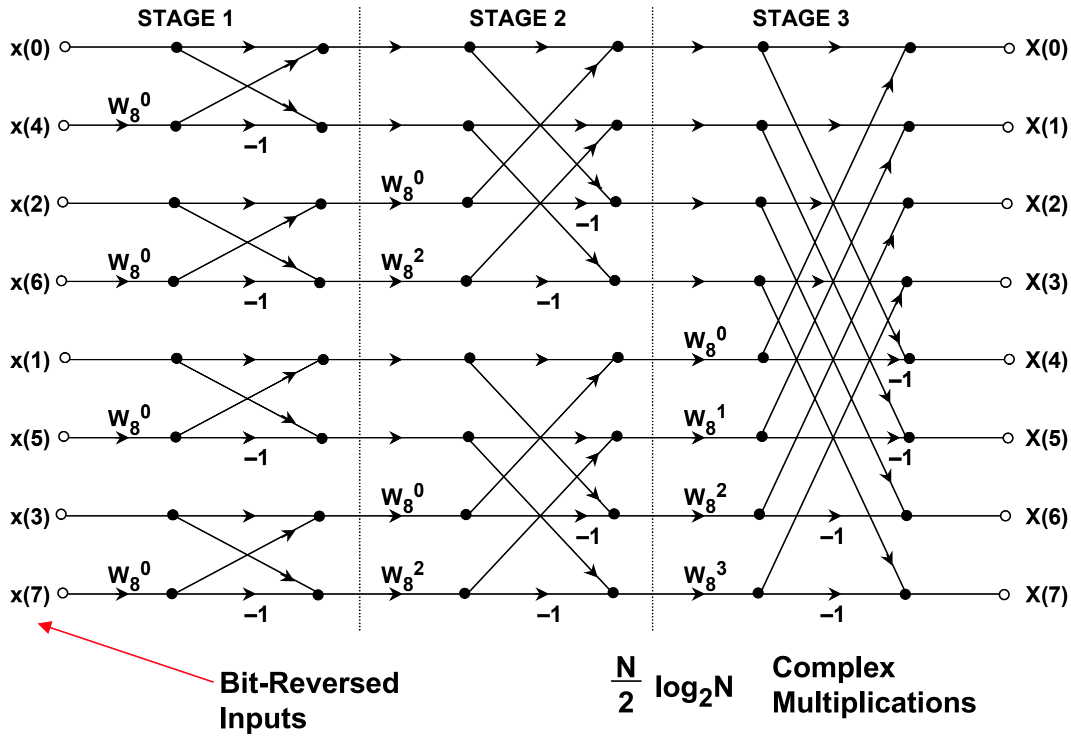

Figure 5.15

The *bit reversal* algorithm used to perform this re-ordering is shown in Figure 5.16. The decimal index, n, is converted to its binary equivalent. The binary bits are then placed in reverse order, and converted back to a decimal number. Bit reversing is often performed in DSP hardware in the data address generator (DAG), thereby simplifying the software, reducing overhead, and speeding up the computations.

The computation of the FFT using *decimation-in-frequency* (DIF) is shown in Figures 5.17 and 5.18. This method requires that the bit reversal algorithm be applied to the output X(k). Note that the butterfly for the DIF algorithm differs slightly from the decimation-in-time butterfly as shown in Figure 5.19.

The use of decimation-in-time versus decimation-in-frequency algorithms is largely a matter of preference, as either yields the same result. System constraints may make one of the two a more optimal solution.

It should be noted that the algorithms required to compute the inverse FFT are nearly identical to those required to compute the FFT, assuming complex FFTs are used. In fact, a useful method for verifying a complex FFT algorithm consists of first taking the FFT of the x(n) time samples and then taking the inverse FFT of the X(k). At the end of this process, the original time samples, Re x(n), should be obtained and the imaginary part, Im x(n), should be zero (within the limits of the mathematical round off errors).

BIT REVERSAL EXAMPLE FOR N = 8

Decimal Number :	0	1	2	3	4	5	6	7
Binary Equivalent :	000	001	010	011	100	101	110	111
Bit-Reversed Binary :	000	100	010	110	001	101	011	111
Decimal Equivalent :	0	4	2	6	1	5	3	7

Figure 5.16

COMPUTATION OF AN 8-POINT DFT IN THREE STAGES USING DECIMATION-IN-FREQUENCY

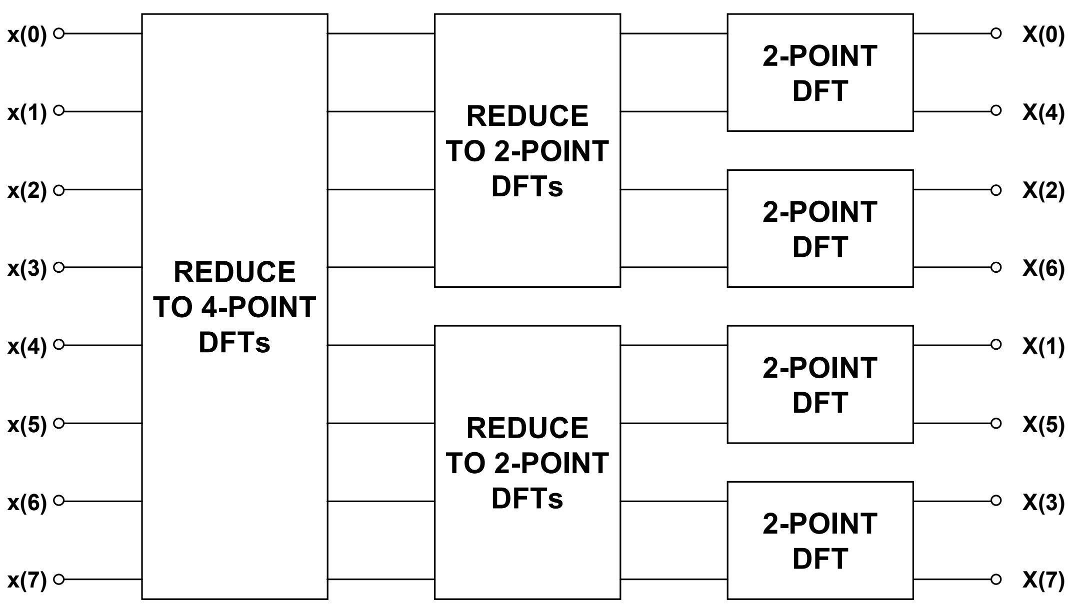

Figure 5.17

EIGHT-POINT DECIMATION-IN-FREQUENCY FFT ALGORITHM

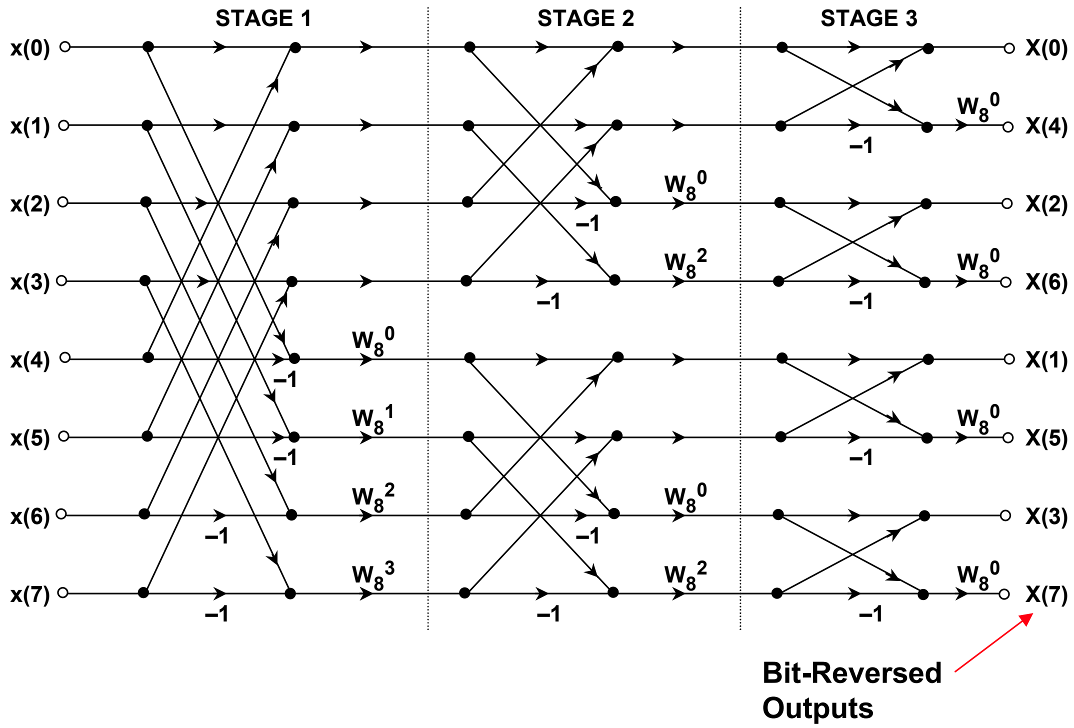

Figure 5.18

THE BASIC BUTTERFLY COMPUTATION IN THE DECIMATION-IN-FREQUENCY FFT ALGORITHM

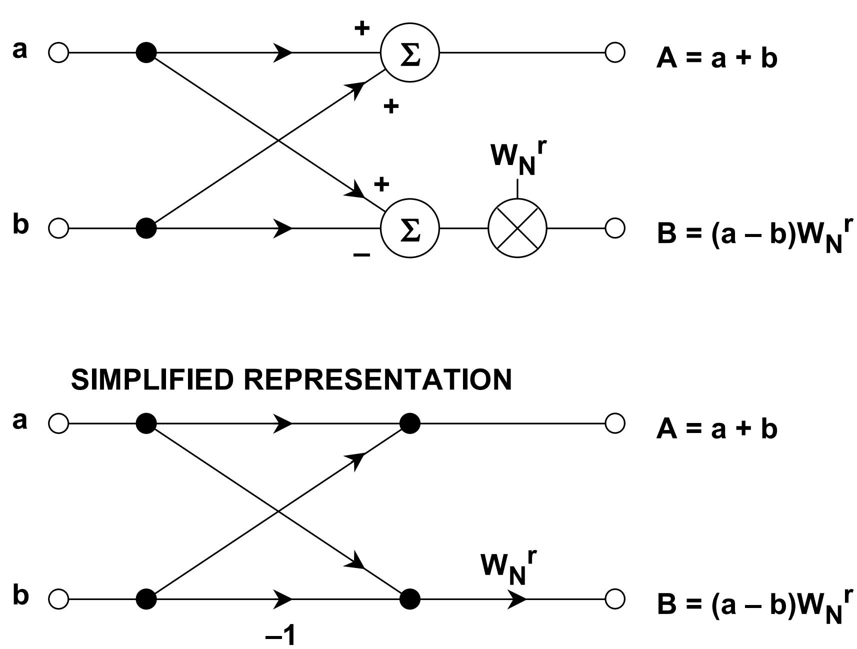

Figure 5.19

The FFTs discussed up to this point are radix-2 FFTs, i.e., the computations are based on 2-point butterflies. This implies that the number of points in the FFT must be a power of 2. If the number of points in an FFT is a power of 4, however, the FFT can be broken down into a number of 4-point DFTs as shown in Figure 5.20. This is called a radix-4 FFT. The fundamental decimation-in-time butterfly for the radix-4 FFT is shown in Figure 5.21.

The radix-4 FFT requires fewer complex multiplications but more additions than the radix-2 FFT for the same number of points. Compared to the radix-2 FFT, the radix-4 FFT trades more complex data addressing and twiddle factors with less computation. The resulting savings in computation time varies between different DSPs but a radix-4 FFT can be as much as twice as fast as a radix-2 FFT for DSPs with optimal architectures.

COMPUTATION OF A 16-POINT DFT IN THREE STAGES USING RADIX-4 DECIMATION-IN-TIME ALGORITHM

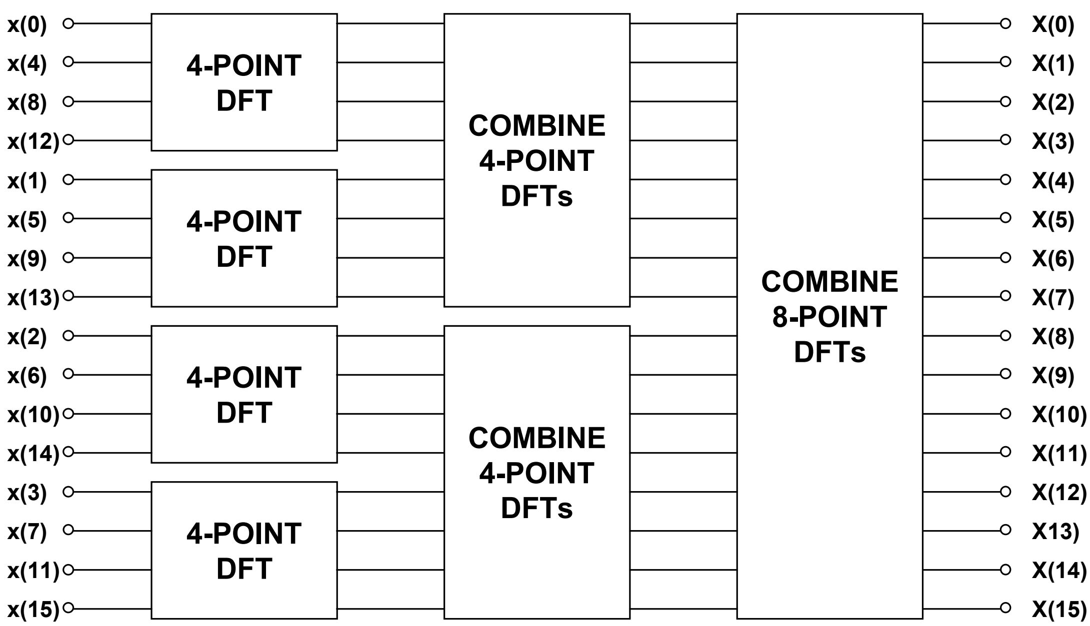

Figure 5.20

RADIX-4 FFT DECIMATION-IN-TIME BUTTERFLY

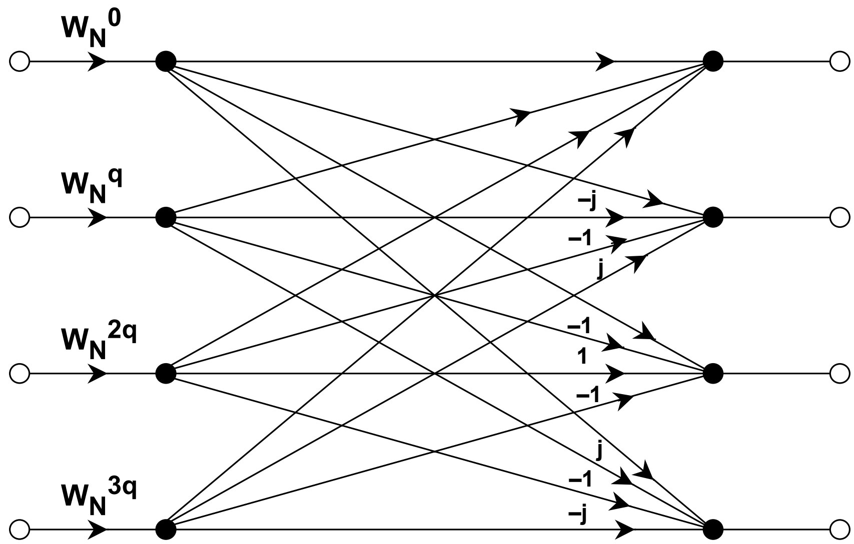

Figure 5.21

FFT HARDWARE IMPLEMENTATION AND BENCHMARKS

In general terms, the memory requirements for an N-point FFT are N locations for the real data, N locations for the imaginary data, and N locations for the sinusoid data (sometimes referred to as twiddle factors). Additional memory locations will be required if windowing is used. Assuming the memory requirements are met, the DSP must perform the necessary calculations in the required time. Many DSP vendors will either give a performance benchmark for a specified FFT size or calculation time for a butterfly. When comparing FFT specifications, it is important to make sure that the same type of FFT is used in all cases. For example, the 1024-point FFT benchmark on one DSP derived from a radix-2 FFT should not be compared with the radix-4 benchmark from another DSP.

Another consideration regarding FFTs is whether to use a fixed-point or a floating-point processor. The results of a butterfly calculation can be larger than the inputs to the butterfly. This data growth can pose a potential problem in a DSP with a fixed number of bits. To prevent data overflow, the data needs to be scaled beforehand leaving enough extra bits for growth. Alternatively, the data can be scaled after each stage of the FFT computation. The technique of scaling data after each pass of the FFT is known as *block floating point*. It is called this because a full array of data is scaled as a block regardless of whether or not each element in the block needs to be scaled. The complete block is scaled so that the relative relationship of each data word remains the same. For example, if each data word is shifted right by one bit (divided by 2), the absolute values have been changed, but relative to each other, the data stays the same.

In a 16-bit fixed-point DSP, a 32-bit word is obtained after multiplication. The Analog Device's 21xx-series DSPs have extended dynamic range by providing a 40-bit internal register in the multiply-accumulator (MAC).

The use of a floating-point DSP eliminates the need for data scaling and therefore results in a simpler FFT routine, however the tradeoff is the increased processing time required for the complex floating-point arithmetic. In addition, a 32-bit floating-point DSP will obviously have less round off noise than a 16-bit fixed-point DSP. Figure 5.22 summarizes the FFT benchmarks for popular Analog Devices' DSPs. Notice in particular that the ADSP-TS001 TigerSHARC™ DSP offers both fixed-point and floating-point modes, thereby providing an exceptional degree of programming flexibility.

RADIX-2 COMPLEX FFT
HARDWARE BENCHMARK COMPARISONS

- **ADSP-2189M, 16-bit, Fixed-Point @ 75MHz**
    - **453µs (1024-Point)**
- **ADSP-21160 SHARC™, 32-bit, Floating-Point @ 100MHz**
    - **180µs (1024-Point), 2 channels, SIMD Mode**
    - **115µs (1024-Point), 1 channel, SIMD Mode**
- **ADSP-TS001 TigerSHARC™ @ 150MHz,**
    - **16-bit, Fixed-Point Mode**
        - **7.3µs (256-Point FFT)**
    - **32-bit, Floating-Point Mode**
        - **69µs (1024-Point)**

Figure 5.22

DSP REQUIREMENTS FOR REAL TIME FFT APPLICATIONS

There are two basic ways to acquire data from a real-world signal, either one sample at a time (continuous processing), or one frame at a time (batch processing). Sample-based systems, like a digital filter, acquire data one sample at a time. For each sample clock, a sample comes into the system, and a processed sample is sent to the output. Frame-based systems, like an FFT-based digital spectrum analyzer, acquire a frame (or block of samples). Processing occurs on the entire frame of data and results in a frame of transformed output data.

In order to maintain real time operation, the entire FFT must therefore be calculated during the frame period. This assumes that the DSP is collecting the data for the next frame while it is calculating the FFT for the current frame of data. Acquiring the data is one area where special architectural features of DSPs come into play. Seamless data acquisition is facilitated by the DSP's flexible data addressing capabilities in conjunction with its direct memory accessing (DMA) channels.

Assume the DSP is the ADSP-TS001 TigerSHARC which can calculate a 1024-point 32-bit complex floating-point FFT in 69µs. The maximum sampling frequency is therefore 1024/69µs = 14.8MSPS. This implies a signal bandwidth of less than 7.4MHz. It is also assumed that there is no additional FFT overhead or data transfer limitation.

REAL-TIME FFT PROCESSING EXAMPLE

- **Assume 69µs Execution Time for Radix-2, 1024-point FFT (TigerSHARC, 32-bit Mode)**
- **f_S (maximum) <** $\dfrac{\textbf{1024 Samples}}{\textbf{69µs}}$ **= 14.8MSPS**
- **Therefore Input Signal Bandwidth < 7.4MHz**
- **This Assumes No Additional FFT Overhead and No Input/Output Data Transfer Limitations**

Figure 5.23

The above example will give an estimate of the maximum bandwidth signal which can be handled by a given DSP using its FFT benchmarks. Another way to approach the issue is to start with the signal bandwidth and develop the DSP requirements. If the signal bandwidth is known, the required sampling frequency can be estimated by multiplying by a factor between 2 and 2.5 (the increased sampling rate may be required to ease the requirements on the antialiasing filter which precedes the ADC). The next step is to determine the required number of points in the FFT to achieve the desired frequency resolution. The frequency resolution is obtained by dividing the sampling rate f_S by N, the number of points in the FFT. These and other FFT considerations are shown in Figure 5.24.

The number of FFT points also determines the noise floor of the FFT with respect to the broadband noise level, and this may also be a consideration. Figure 5.25 shows the relationships between the system fullscale signal level, the broadband noise level (measured over the bandwidth DC to $f_S/2$), and the FFT noise floor. Notice that the FFT processing gain is determined by the number of points in the FFT. The FFT acts like an analog spectrum analyzer with a sweep bandwidth of f_S/N. Increasing the number of points increases the FFT resolution and narrows its bandwidth, thereby reducing the noise floor. This analysis neglects noise caused by the FFT round off error. In practice, the ADC which is used to digitize the signal produces quantization noise which is the dominant noise source.

At this point it is time to examine actual DSPs and their FFT processing times to make sure real time operation can be achieved. This means that the FFT must be calculated during the acquisition time for one frame of data which is N/f_S. Other considerations such as fixed-point vs. floating-point, radix-2 vs. radix-4, and processor power dissipation and cost may be other considerations.

REAL TIME FFT CONSIDERATIONS

- **Signal Bandwidth**
- **Sampling Frequency, f_s**
- **Number of Points in FFT, N**
- **Frequency Resolution = f_s / N**
- **Maximum Time to Calculate N-Point FFT = N / f_s**
- **Fixed-Point vs. Floating Point DSP**
- **Radix-2 vs. Radix-4 Execution Time**
- **FFT Processing Gain = 10 $\log_{10}$(N / 2)**
- **Windowing Requirements**

Figure 5.24

FFT PROCESSING GAIN
NEGLECTING ROUND OFF ERROR

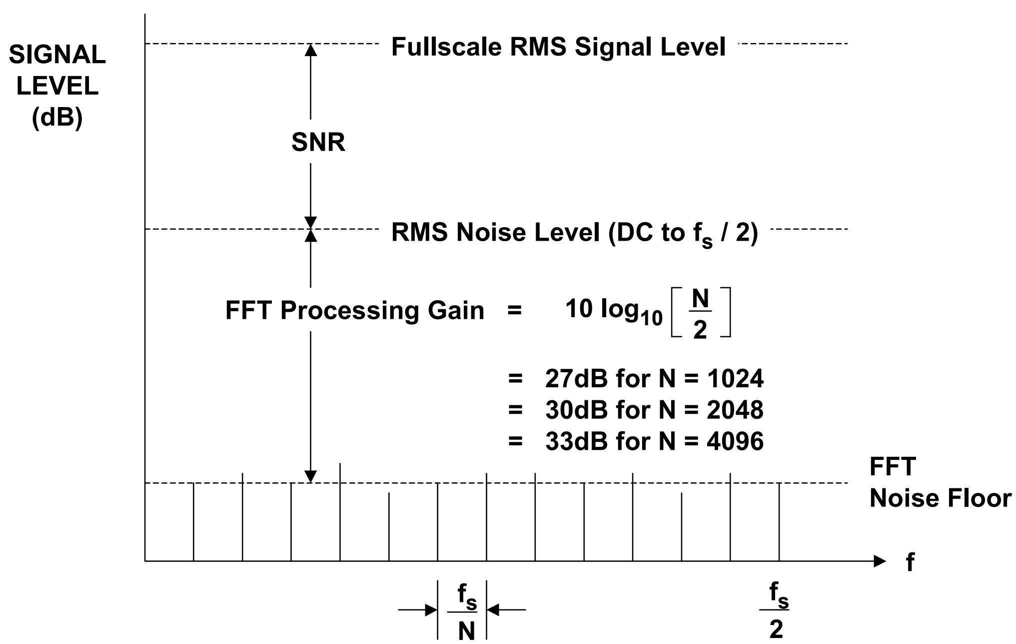

Figure 5.25

SPECTRAL LEAKAGE AND WINDOWING

Spectral leakage in FFT processing can best be understood by considering the case of performing an N-point FFT on a pure sinusoidal input. Two conditions will be considered. In Figure 5.26, the ratio between the sampling frequency and the input sinewave frequency is such that precisely an integral number of cycles are contained within the data window (frame, or record). Recall that the DFT assumes that an infinite number of these windows are placed end-to-end to form a periodic waveform as shown in the diagram as the periodic extensions. Under these conditions, the waveform appears continuous (no discontinuities), and the DFT or FFT output will be a single tone located at the input signal frequency.

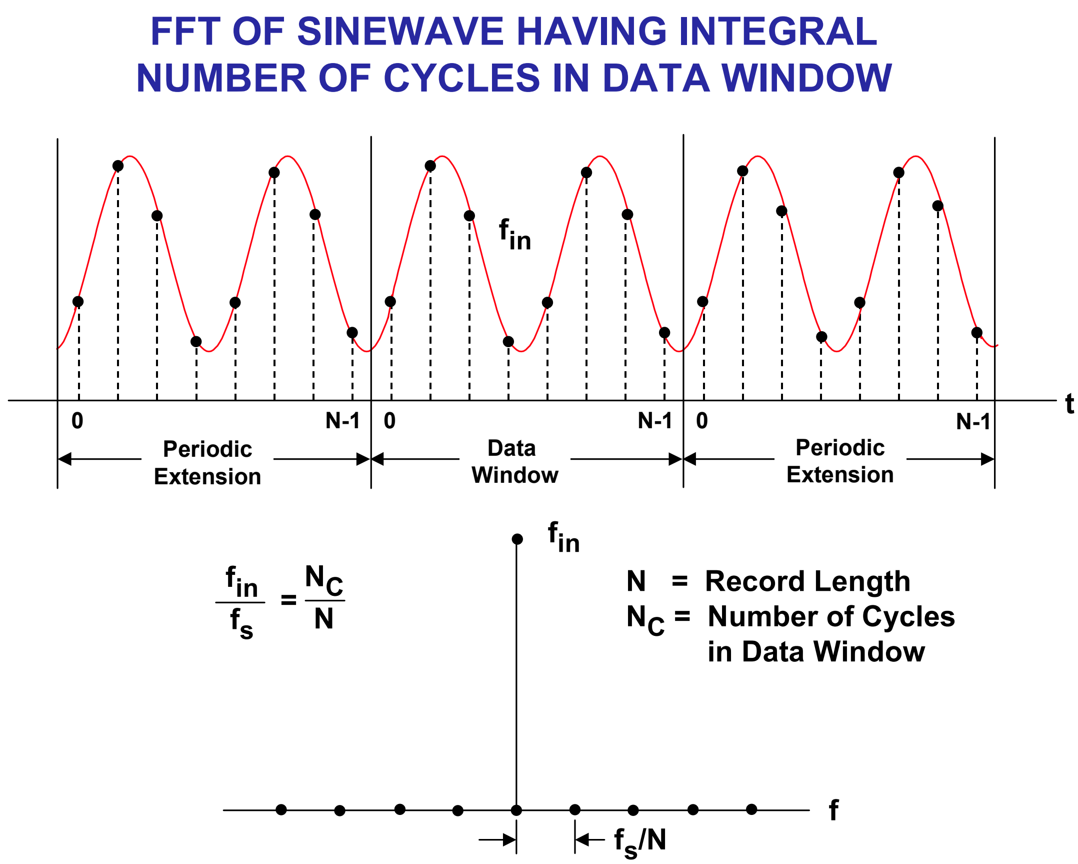

Figure 5.26

Figure 5.27 shows the condition where there are not an integral number of sinewave cycles within the data window. The discontinuities which occur at the endpoints of the data window result in leakage in the frequency domain because of the harmonics which are generated. In addition to the sidelobes, the main lobe of the sinewave is smeared over several frequency bins. This process is equivalent to multiplying the input sinewave by a rectangular window pulse which has the familiar sin(x)/x frequency response and associated smearing and sidelobes.

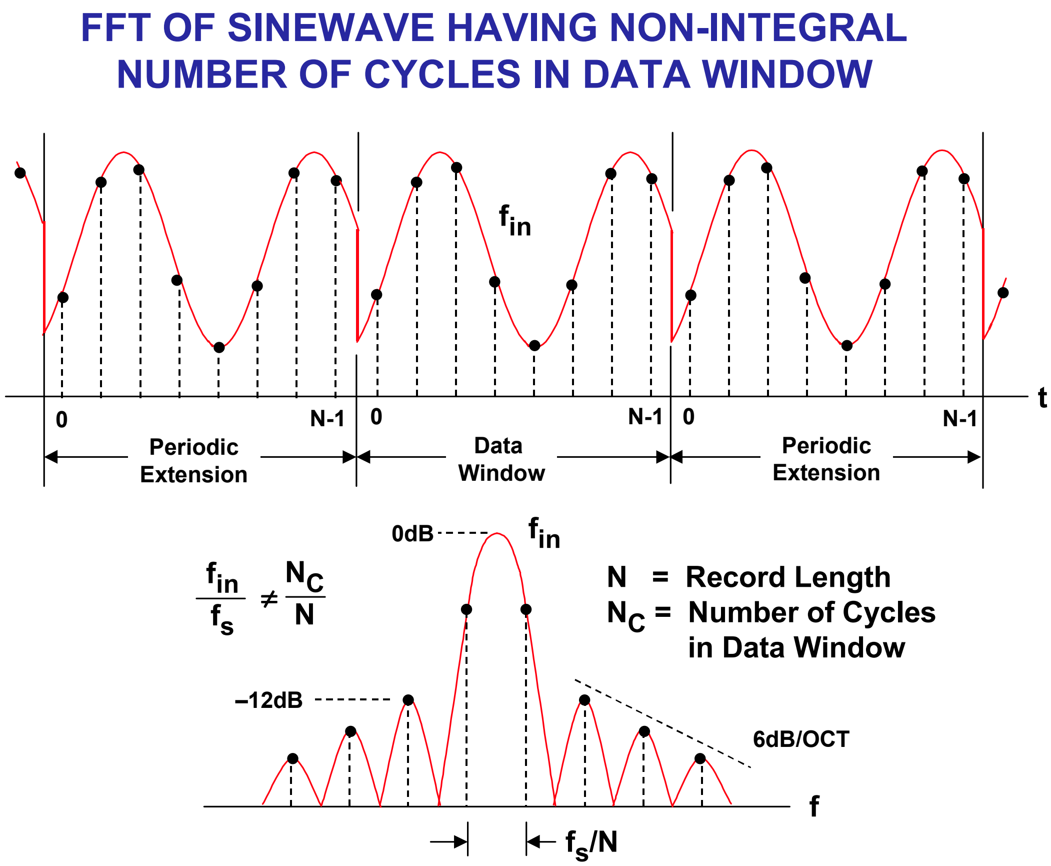

Figure 5.27

Notice that the first sidelobe is only 12dB below the fundamental, and that the sidelobes roll off at only 6dB/octave. This situation would be unsuitable for most spectral analysis applications. Since in practical FFT spectral analysis applications the exact input frequencies are unknown, something must be done to minimize these sidelobes. This is done by choosing a window function other than the rectangular window. The input time samples are multiplied by an appropriate window function which brings the signal to zero at the edges of the window as shown in Figure 5.28. The selection of a window function is primarily a tradeoff between main lobe spreading and sidelobe roll off. Reference 7 is highly recommended for an in-depth treatment of windows.

The mathematical functions which describe four popular window functions (Hamming, Blackman, Hanning, and Minimum 4-term Blackman-Harris) are shown in Figure 5.29. The computations are straightforward, and the window function data points are usually precalculated and stored in the DSP memory to minimize their impact on FFT processing time. The frequency response of the rectangular, Hamming, and Blackman windows are shown in Figure 5.30. Figure 5.31 shows the tradeoff between main lobe spreading and sidelobe amplitude and roll off for the popular window functions.

WINDOWING TO REDUCE SPECTRAL LEAKAGE

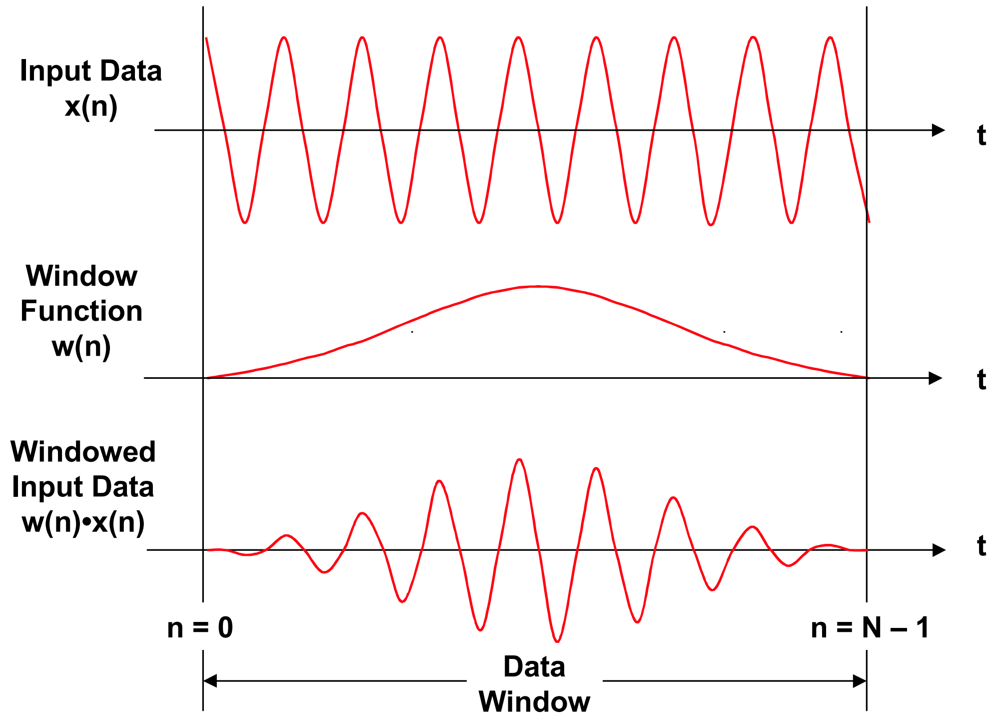

Figure 5.28

SOME POPULAR WINDOW FUNCTIONS

- **Hamming:** $w(n) = 0.54 - 0.46 \cos\left[\frac{2\pi n}{N}\right]$
- **Blackman:** $w(n) = 0.42 - 0.5 \cos\left[\frac{2\pi n}{N}\right] + 0.08 \cos\left[\frac{4\pi n}{N}\right]$
- **Hanning:** $w(n) = 0.5 - 0.5 \cos\left[\frac{2\pi n}{N}\right]$
- **Minimum 4-Term Blackman Harris** $w(n) = 0.35875 - 0.48829 \cos\left[\frac{2\pi n}{N}\right] + 0.14128 \cos\left[\frac{4\pi n}{N}\right] - 0.01168 \cos\left[\frac{6\pi n}{N}\right]$

$\boxed{0 \leq n \leq N-1}$

Figure 5.29

FREQUENCY RESPONSE OF RECTANGULAR, HAMMING, AND BLACKMAN WINDOWS FOR N = 256

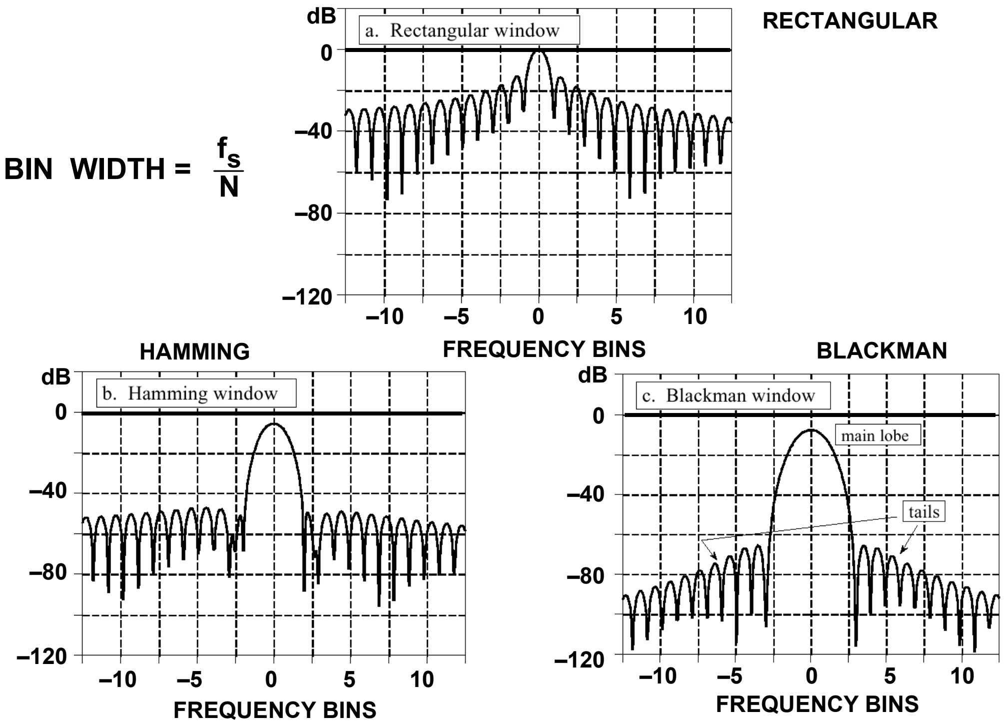

Figure 5.30

POPULAR WINDOWS AND FIGURES OF MERIT

WINDOW FUNCTION	3dB BW (Bins)	6dB BW (Bins)	HIGHEST SIDELOBE (dB)	SIDELOBE ROLLOFF (dB/Octave)
Rectangle	0.89	1.21	–12	6
Hamming	1.3	1.81	– 43	6
Blackman	1.68	2.35	–58	18
Hanning	1.44	2.00	–32	18
Minimum 4-Term Blackman-Harris	1.90	2.72	–92	6

Figure 5.31

REFERENCES

1. Steven W. Smith, **The Scientist and Engineer's Guide to Digital Signal Processing**, Second Edition, 1999, California Technical Publishing, P.O. Box 502407, San Diego, CA 92150. Also available for free download at: http://www.dspguide.com or http://www.analog.com

2. C. Britton Rorabaugh, **DSP Primer**, McGraw-Hill, 1999.

3. Richard J. Higgins, **Digital Signal Processing in VLSI,** Prentice-Hall, 1990.

4. A. V. Oppenheim and R. W. Schafer, **Digital Signal Processing**, Prentice-Hall, 1975.

5. L. R. Rabiner and B. Gold, **Theory and Application of Digital Signal Processing**, Prentice-Hall, 1975.

6. John G. Proakis and Dimitris G. Manolakis, **Introduction to Digital Signal Processing**, MacMillian, 1988.

7. Fredrick J. Harris, *On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform*, **Proc. IEEE**, Vol. 66, No. 1, 1978 pp. 51-83.

8. R. W. Ramirez, **The FFT: Fundamentals and Concepts**, Prentice-Hall, 1985.

9. J. W. Cooley and J. W. Tukey, *An Algorithm for the Machine Computation of Complex Fourier Series*, **Mathematics Computation, Vol. 19**, pp. 297-301, April 1965.

10. **Digital Signal Processing Applications Using the ADSP-2100 Family**, Vol. 1 and Vol. 2, Analog Devices, Free Download at: http://www.analog.com

11. **ADSP-21000 Family Application Handbook**, Vol. 1, Analog Devices, Free Download at: http://www.analog.com