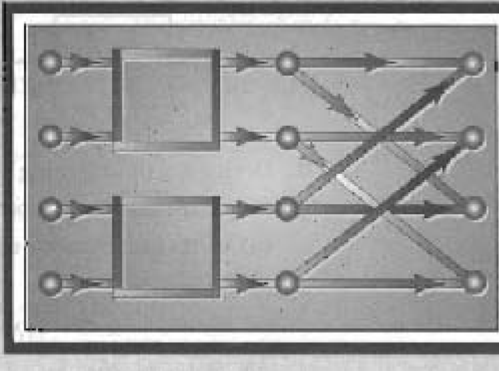
9 Computation of the Discrete Fourier Transform



9.0 INTRODUCTION

The discrete Fourier transform (DFT) plays an important role in the analysis, design, and implementation of discrete-time signal-processing algorithms and systems because the basic properties of the discrete-time Fourier transform and discrete Fourier transform, discussed in Chapters 2 and 8, respectively, make it particularly convenient to analyze and design systems in the Fourier domain. It is equally important that efficient algorithms exist for explicitly computing the DFT. As a result, the DFT is an important component in many practical applications of discrete-time systems.

In this chapter, we discuss several methods for computing values of the DFT. The major focus of the chapter is a particularly efficient class of algorithms for the digital computation of the N-point DFT. Collectively, these efficient algorithms, which are discussed in Sections 9.2, 9.3, and 9.5, are called FFT algorithms. To achieve the highest efficiency, the FFT algorithms must compute all N values of the DFT. When we require values of the DFT at only a few frequencies in the range $0 \leq \omega < 2\pi$, other algorithms may be more efficient and flexible, even though they are less efficient than the FFT algorithms for computation of all the values of the DFT. Examples of such algorithms are the Goertzel algorithm, discussed in Section 9.1.2, and the chirp transform algorithm, discussed in Section 9.6.2.

There are many ways to measure the complexity and efficiency of an implementation or algorithm, and a final assessment depends on both the available implementation technology and the intended application. We will use the number of arithmetic multiplications and additions as a measure of computational complexity. This measure is simple to apply, and the number of multiplications and additions is directly related to the computational speed when algorithms are implemented on general-purpose digital computers or special-purpose processors. However, other measures are sometimes more appropriate. For example, in custom VLSI implementations, the area of the chip and power requirements are important considerations and may not be directly related to the number of arithmetic operations.

In terms of multiplications and additions, the class of FFT algorithms can be orders of magnitude more efficient than competing algorithms. The efficiency of FFT algorithms is so high, in fact, that in many cases the most efficient procedure for implementing a convolution is to compute the transform of the sequences to be convolved, multiply their transforms, and then compute the inverse transform of the product of transforms. The details of this technique were discussed in Section 8.7. In seeming contradiction to this, there is a set of algorithms (mentioned briefly in Section 9.6) for evaluation of the DFT (or a more general set of samples of the Fourier transform) that derive their efficiency from a reformulation of the Fourier transform in terms of a convolution and thereby implement the Fourier transform computation by using efficient procedures for evaluating the associated convolution. This suggests the possibility of implementing a convolution by multiplication of DFTs, where the DFTs have been implemented by first expressing them as convolutions and then taking advantage of efficient procedures for implementing the associated convolutions. While this seems on the surface to be a basic contradiction, we will see in Section 9.6 that in certain cases it is an entirely reasonable approach.

In the sections that follow, we consider a number of algorithms for computing the discrete Fourier transform. We begin in Section 9.1 with discussions of direct computation methods, i.e., methods based on direct use of the defining relation for the DFT as a computational formula. We include in this discussion the Goertzel algorithm (Goertzel, 1958), which requires computation proportional to N^2, but with a smaller constant of proportionality than that of the direct evaluation of the defining formula. One of the principal advantages of the direct evaluation method or the Goertzel algorithm is that they are not restricted to computation of the DFT, but can be used to compute any desired set of samples of the DTFT of a finite-length sequence.

In Sections 9.2 and 9.3 we present a detailed discussion of FFT algorithms for which computation is proportional to $N \log_2 N$. This class of algorithms is considerably more efficient in terms of arithmetic operations than the Goertzel algorithm, but is specifically oriented toward computation of *all* the values of the DFT. We do not attempt to be exhaustive in our coverage of that class of algorithms, but we illustrate the general principles common to all algorithms of this type by considering in detail only a few of the more commonly used schemes.

In Section 9.4, we consider some of the practical issues that arise in implementing the power-of-two-length FFT algorithms discussed in Sections 9.2 and 9.3. Section 9.5 provides a brief overview of algorithms for N a composite number including a brief reference to FFT algorithms that are optimized for a particular computer architecture. In Section 9.6, we discuss algorithms that rely on formulating the computation of the DFT in terms of a convolution. In Section 9.7, we consider effects of arithmetic round-off in FFT algorithms.

9.1 DIRECT COMPUTATION OF THE DISCRETE FOURIER TRANSFORM

As defined in Chapter 8, the DFT of a finite-length sequence of length N is

$$X[k] = \sum_{n=0}^{N-1} x[n]W_N^{kn}, \qquad k = 0, 1, \ldots, N-1, \tag{9.1}$$

where $W_N = e^{-j(2\pi/N)}$. The inverse discrete Fourier transform is given by

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]W_N^{-kn}, \qquad n = 0, 1, \ldots, N-1. \tag{9.2}$$

In Eqs. (9.1) and (9.2), both $x[n]$ and $X[k]$ may be complex.[1] Since the expressions on the right-hand sides of those equations differ only in the sign of the exponent of W_N and in the scale factor $1/N$, a discussion of computational procedures for Eq. (9.1) applies with straightforward modifications to Eq. (9.2). (See Problem 9.1.)

Most approaches to improving the efficiency of computation of the DFT exploit the symmetry and periodicity properties of W_N^{kn}; specifically,

$$W_N^{k(N-n)} = W_N^{-kn} = (W_N^{kn})^* \text{ (complex conjugate symmetry)} \tag{9.3a}$$

$$W_N^{kn} = W_N^{k(n+N)} = W_N^{(k+N)n} \qquad \text{(periodicity in } n \text{ and } k\text{).} \tag{9.3b}$$

(Since $W_N^{kn} = \cos(2\pi kn/N) - j\sin(2\pi kn/N)$ these properties are a direct consequence of the symmetry and periodicity of the underlying sine and cosine functions.) Because the complex numbers W_N^{kn} have the role of coefficients in Eqs. (9.1) and (9.2), the redundancy implied by these conditions can be used to advantage in reducing the amount of computation required for their evaluation.

9.1.1 Direct Evaluation of the Definition of the DFT

To create a frame of reference, consider first the direct evaluation of the defining DFT expression in Eq. (9.1). Since $x[n]$ may be complex, N complex multiplications and $(N-1)$ complex additions are required to compute each value of the DFT if we use Eq. (9.1) directly as a formula for computation. To compute all N values therefore requires a total of N^2 complex multiplications and $N(N-1)$ complex additions. Expressing

[1]In discussing algorithms for computing the DFT of a finite-length sequence $x[n]$, it is worthwhile to recall from Chapter 8 that the DFT values defined by Eq. (9.1) can be thought of either as samples of the DTFT $X(e^{j\omega})$ at frequencies $\omega_k = 2\pi k/N$ or as coefficients in the discrete-time Fourier series for the periodic sequence

$$\tilde{x}[n] = \sum_{r=-\infty}^{\infty} x[n+rN].$$

It will be helpful to keep both interpretations in mind and to be able to switch focus from one to the other as is convenient.

Eq. (9.1) in terms of operations on real numbers, we obtain

$$X[k] = \sum_{n=0}^{N-1} \Big[(\mathcal{R}e\{x[n]\}\mathcal{R}e\{W_N^{kn}\} - \mathcal{I}m\{x[n]\}\mathcal{I}m\{W_N^{kn}\}) + j(\mathcal{R}e\{x[n]\}\mathcal{I}m\{W_N^{kn}\} + \mathcal{I}m\{x[n]\}\mathcal{R}e\{W_N^{kn}\}) \Big], \quad k = 0, 1, \dots, N-1, \tag{9.4}$$

which shows that each complex multiplication $x[n] \cdot W_N^{kn}$ requires four real multiplications and two real additions, and each complex addition requires two real additions. Therefore, for each value of k, the direct computation of $X[k]$ requires $4N$ real multiplications and $(4N-2)$ real additions.[2] Since $X[k]$ must be computed for N different values of k, the direct computation of the discrete Fourier transform of a sequence $x[n]$ requires $4N^2$ real multiplications and $N(4N-2)$ real additions. Besides the multiplications and additions called for by Eq. (9.4), the digital computation of the DFT on a general-purpose digital computer or with special-purpose hardware also requires provision for storing and accessing the N complex input sequence values $x[n]$ and values of the complex coefficients W_N^{kn}. Since the amount of computation, and thus the computation time, is approximately proportional to N^2, it is evident that the number of arithmetic operations required to compute the DFT by the direct method becomes very large for large values of N. For this reason, we are interested in computational procedures that reduce the number of multiplications and additions.

As an illustration of how the properties of W_N^{kn} can be exploited, using the symmetry property in Eq. (9.3a), we can group terms in the summation in Eq. (9.4) for n and $(N-n)$. For example, the grouping

$$\mathcal{R}e\{x[n]\}\mathcal{R}e\{W_N^{kn}\} + \mathcal{R}e\{x[N-n]\}\mathcal{R}e\{W_N^{k(N-n)}\} = (\mathcal{R}e\{x[n]\} + \mathcal{R}e\{x[N-n]\})\mathcal{R}e\{W_N^{kn}\}$$

eliminates one real multiplication, as does the grouping

$$-\mathcal{I}m\{x[n]\}\mathcal{I}m\{W_N^{kn}\} - \mathcal{I}m\{x[N-n]\}\mathcal{I}m\{W_N^{k(N-n)}\} = -(\mathcal{I}m\{x[n]\} - \mathcal{I}m\{x[N-n]\})\mathcal{I}m\{W_N^{kn}\}.$$

Similar groupings can be used for the other terms in Eq. (9.4). In this way, the number of multiplications can be reduced by approximately a factor of 2. We can also take advantage of the fact that for certain values of the product kn, the implicit sine and cosine functions take on the value 1 or 0, thereby eliminating the need for multiplications. However, reductions of this type still leave us with an amount of computation that is proportional to N^2. Fortunately, the second property [Eq. (9.3b)], the periodicity of the complex sequence W_N^{kn}, can be exploited with recursion to achieve significantly greater reductions of the computation.

9.1.2 The Goertzel Algorithm

The Goertzel algorithm (Goertzel, 1958) is an example of how the periodicity of the sequence W_N^{kn} can be used to reduce computation. To derive the algorithm, we begin

[2] Throughout the discussion, the formula for the number of computations may be only approximate. Multiplication by W_N^0, for example, does not require a multiplication. Nevertheless, when N is large, the estimate of computational complexity obtained by including such multiplications is sufficiently accurate to permit comparisons between different classes of algorithms.

by noting that

$$W_N^{-kN} = e^{j(2\pi/N)Nk} = e^{j2\pi k} = 1, \tag{9.5}$$

since k is an integer. This is a result of the periodicity with period N of W_N^{-kn} in either n or k. Because of Eq. (9.5), we may multiply the right side of Eq. (9.1) by W_N^{-kN} without affecting the equation. Thus,

$$X[k] = W_N^{-kN} \sum_{r=0}^{N-1} x[r] W_N^{kr} = \sum_{r=0}^{N-1} x[r] W_N^{-k(N-r)}. \tag{9.6}$$

To suggest the final result, we define the sequence

$$y_k[n] = \sum_{r=-\infty}^{\infty} x[r] W_N^{-k(n-r)} u[n-r]. \tag{9.7}$$

From Eqs. (9.6) and (9.7) and the fact that $x[n] = 0$ for $n < 0$ and $n \geq N$, it follows that

$$X[k] = y_k[n]\Big|_{n=N}. \tag{9.8}$$

Equation (9.7) can be interpreted as a discrete convolution of the finite-duration sequence $x[n]$, $0 \leq n \leq N-1$, with the sequence $W_N^{-kn}u[n]$. Consequently, $y_k[n]$ can be viewed as the response of a system with impulse response $W_N^{-kn}u[n]$ to a finite-length input $x[n]$. In particular, $X[k]$ is the value of the output when $n = N$.

The signal flow graph of a system with impulse response $W_N^{-kn}u[n]$ is shown in Figure 9.1, which represents the difference equation

$$y_k[n] = W_N^{-k} y_k[n-1] + x[n], \tag{9.9}$$

where initial rest conditions are assumed. Since the general input $x[n]$ and the coefficient W_N^{-k} are both complex, the computation of each new value of $y_k[n]$ using the system of Figure 9.1 requires 4 real multiplications and 4 real additions. All the intervening values $y_k[1], y_k[2], \ldots, y_k[N-1]$ must be computed in order to compute $y_k[N] = X[k]$, so the use of the system in Figure 9.1 as a computational algorithm requires $4N$ real multiplications and $4N$ real additions to compute $X[k]$ for a particular value of k. Thus, this procedure is slightly less efficient than the direct method. However, it avoids the computation or storage of the coefficients W_N^{kn}, since these quantities are implicitly computed by the recursion implied by Figure 9.1.

It is possible to retain this simplification while reducing the number of multiplications by a factor of 2. To see how this may be done, note that the system function of the system of Figure 9.1 is

$$H_k(z) = \frac{1}{1 - W_N^{-k} z^{-1}}. \tag{9.10}$$

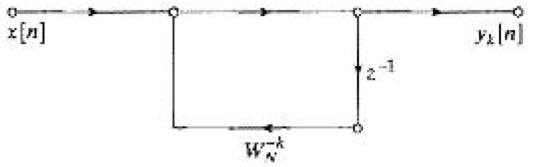


Figure 9.1 Flow graph of 1st-order complex recursive computation of $X[k]$.

Multiplying both the numerator and the denominator of $H_k(z)$ by the factor $(1 - W_N^k z^{-1})$, we obtain

$$\begin{aligned} H_k(z) &= \frac{1 - W_N^k z^{-1}}{(1 - W_N^{-k} z^{-1})(1 - W_N^k z^{-1})} \\ &= \frac{1 - W_N^k z^{-1}}{1 - 2\cos(2\pi k/N) z^{-1} + z^{-2}}. \end{aligned} \tag{9.11}$$

The signal flow graph of Figure 9.2 corresponds to the direct form II implementation of the system function of Eq. (9.11) for which the difference equation for the poles is

$$v_k[n] = 2\cos(2\pi k/N) v_k[n-1] - v_k[n-2] + x[n]. \tag{9.12a}$$

After N iterations of Eq. (9.12a) starting with initial rest conditions $w_k[-2] = w_k[-1] = 0$, the desired DFT value can be obtained by implementing the zero as in

$$X[k] = y_k[n]\Big|_{n=N} = v_k[N] - W_N^k v_k[N-1]. \tag{9.12b}$$

If the input is complex, only two real multiplications per sample are required to implement the poles of this system, since the coefficients are real and the factor -1 need not be counted as a multiplication. As in the case of the 1st-order system, for a complex input, four real additions per sample are required to implement the poles (if the input is complex). Since we only need to bring the system to a state from which $y_k[N]$ can be computed, the complex multiplication by $-W_N^k$ required to implement the zero of the system function need not be performed at every iteration of the difference equation, but only after the N^{th} iteration. Thus, the total computation is $2N$ real multiplications and $4N$ real additions for the poles.[3] plus 4 real multiplications and 4 real additions for the zero. The total computation is therefore $2(N+2)$ real multiplications and $4(N+1)$ real additions, about half the number of real multiplications required with the direct method. In this more efficient scheme, we still have the advantage that $\cos(2\pi k/N)$ and W_N^k are the only coefficients that must be computed and stored. The coefficients W_N^{kn} are again computed implicitly in the iteration of the recursion formula implied by Figure 9.2.

As an additional advantage of the use of this network, let us consider the computation of the DFT of $x[n]$ at the two symmetric frequencies $2\pi k/N$ and $2\pi(N-k)/N$,

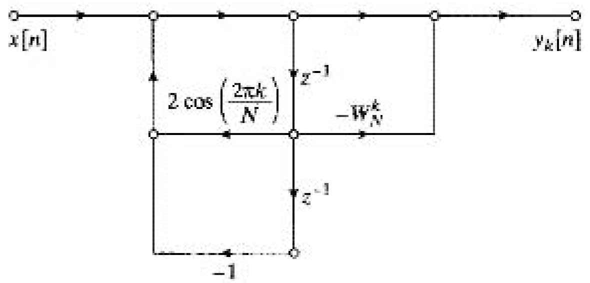


Figure 9.2 Flow graph of 2nd-order recursive computation of $X[k]$ (Goertzel algorithm).

[3]This assumes that $x[n]$ is complex. If $x[n]$ is real, the operation count is N real multiplications and $2N$ real additions for implementing the poles.

that is, the computation of $X[k]$ and $X[N-k]$. It is straightforward to verify that the network in the form of Figure 9.2 required to compute $X[N-k]$ has exactly the same poles as that in Figure 9.2, but the coefficient for the zero is the complex conjugate of that in Figure 9.2. (See Problem 9.21.) Since the zero is implemented only on the final iteration, the $2N$ multiplications and $4N$ additions required for the poles can be used for the computation of two DFT values. Thus, for the computation of all N values of the discrete Fourier transform using the Goertzel algorithm, the number of real multiplications required is approximately N^2 and the number of real additions is approximately $2N^2$. While this is more efficient than the direct computation of the discrete Fourier transform, the amount of computation is still proportional to N^2.

In either the direct method or the Goertzel algorithm we do not need to evaluate $X[k]$ at all N values of k. Indeed, we can evaluate $X[k]$ for any M values of k, with each DFT value being computed by a recursive system of the form of Figure 9.2 with appropriate coefficients. In this case, the total computation is proportional to NM. The Goertzel method and the direct method are attractive when M is small; however, as indicated previously, algorithms are available for which the computation is proportional to $N\log_2 N$ when N is a power of 2. Therefore, when M is less than $\log_2 N$, either the Goertzel algorithm or direct evaluation of the DFT may in fact be the most efficient method, but when all N values of $X[k]$ are required, the decimation-in-time algorithms, to be considered next, are roughly $(N/\log_2 N)$ times more efficient than either the direct method or the Goertzel algorithm.

As we have derived it, the Goertzel algorithm computes the DFT value $X[k]$, which is identical to the DTFT $X(e^{j\omega})$ evaluated at frequency $\omega = 2\pi k/N$. With only a minor modification of the above derivation, we can show that $X(e^{j\omega})$ can be evaluated at any frequency ω_a by iterating the difference equation

$$v_a[n] = 2\cos(\omega_0)v_a[n-1] - v_a[n-2] + x[n]. \tag{9.13a}$$

N times with the desired value of the DTFT obtained by

$$X(e^{j\omega_a}) = e^{-j\omega_a N}(v_a[N] - e^{-j\omega_a}v_a[N-1]). \tag{9.13b}$$

Note that in the case $\omega_a = 2\pi k/N$ Eqs. (9.13a) and (9.13b) reduce to Eqs. (9.12a) and (9.12b). Because Eq. (9.13b) must only be computed once, it is only slightly less efficient to compute the value of $X(e^{j\omega})$ at an arbitrarily chosen frequency than at a DFT frequency.

Still another advantage of the Goertzel algorithm in some real-time applications is that the computation can begin as soon as the first input sample is available. The computation then involves iterating the difference equation Eq. (9.12a) or Eq. (9.13a) as each new input sample becomes available. After N iterations, the desired value of $X(e^{j\omega})$ can be computed with either Eq. (9.12b) or Eq. (9.13b) as is appropriate.

9.1.3 Exploiting both Symmetry and Periodicity

Computational algorithms that exploit both the symmetry and the periodicity of the sequence W_N^{kn} were known long before the era of high-speed digital computation. At that time, any scheme that reduced manual computation by even a factor of 2 was welcomed. Heideman, Johnson and Burrus (1984) have traced the origins of the basic principles of the FFT back to Gauss, as early as 1805. Runge (1905) and later Danielson and Lanczos (1942) described algorithms for which computation was roughly proportional to $N \log_2 N$ rather than N^2. However, the distinction was not of great importance for the small values of N that were feasible for hand computation. The possibility of greatly reduced computation was generally overlooked until about 1965, when Cooley and Tukey (1965) published an algorithm for the computation of the discrete Fourier transform that is applicable when N is a composite number, i.e., the product of two or more integers. The publication of their paper touched off a flurry of activity in the application of the discrete Fourier transform to signal processing and resulted in the discovery of a number of highly efficient computational algorithms. Collectively, the entire set of such algorithms has come to be known as the *fast Fourier transform*, or the FFT.[4]

In contrast to the direct methods discussed above, FFT algorithms are based on the fundamental principle of decomposing the computation of the discrete Fourier transform of a sequence of length N into smaller-length discrete Fourier transforms that are combined to form the N-point transform. These smaller-length transforms may be evaluated by direct methods, or they may be further decomposed into even smaller transforms. The manner in which this principle is implemented leads to a variety of different algorithms, all with comparable improvements in computational speed. In this chapter, we are concerned with two basic classes of FFT algorithms. The first class, called decimation in time, derives its name from the fact that in the process of arranging the computation into smaller transformations, the sequence $x[n]$ (generally thought of as a time sequence) is decomposed into successively smaller subsequences. In the second general class of algorithms, the sequence of discrete Fourier transform coefficients $X[k]$ is decomposed into smaller subsequences—hence its name, decimation in frequency.

We discuss decimation-in-time algorithms in Section 9.2. Decimation-in-frequency algorithms are discussed in Section 9.3. This is an arbitrary ordering. The two sections are essentially independent and can therefore be read in either order.

9.2 DECIMATION-IN-TIME FFT ALGORITHMS

Dramatic efficiency in computing the DFT results from decomposing the computation into successively smaller DFT computations while exploiting both the symmetry and the periodicity of the complex exponential $W_N^{kn} = e^{-j(2\pi/N)kn}$. Algorithms in which the decomposition is based on decomposing the sequence $x[n]$ into successively smaller subsequences are called *decimation-in-time algorithms*.

The principle of decimation-in-time is conveniently illustrated by considering the special case of N an integer power of 2, i.e., $N = 2^\nu$. Since N is divisible by two, we can consider computing $X[k]$ by separating $x[n]$ into two $(N/2)$-point[5] sequences consisting of the even-numbered points $g[n] = x[2n]$ and the odd-numbered points $h[n] = x[2n+1]$. Figure 9.3 shows this decomposition and also the (somewhat obvious, but crucial) fact that the original sequence can be recovered simply by re-interleaving the two sequences.

[4]See Cooley, Lewis and Welch (1967) and Heideman, Johnson and Burrus (1984) for historical summaries of algorithmic developments related to the FFT.

[5]When discussing FFT algorithms, it is common to use the words *sample* and *point* interchangeably to mean *sequence value*, i.e., a single number. Also, we refer to a sequence of length N as an N-point sequence, and the DFT of a sequence of length N will be called an N-point DFT.

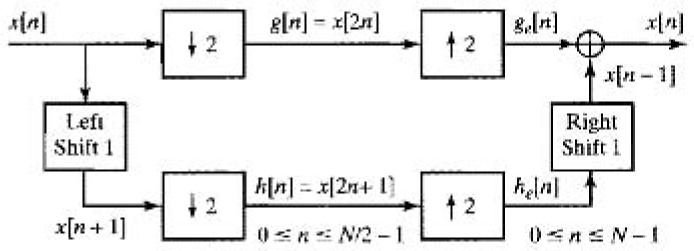


Figure 9.3 Illustration of the basic principle of decimation-in-time.

To understand the significance of Figure 9.3 as an organizing principle for computing the DFT, it is helpful to consider the frequency-domain equivalents of the operations depicted in the block diagram. First, note that the time-domain operation labeled "Left Shift 1" corresponds in the frequency domain to multiplying $X(e^{j\omega})$ by $e^{j\omega}$. As discussed in Section 4.6.1, corresponding to the compression of the time sequences by 2, the DTFTs $G(e^{j\omega})$ and $H(e^{j\omega})$ (and therefore $G[k]$ and $H[k]$) are obtained by frequency-domain aliasing that occurs after expanding the frequency scale by the substitution $\omega \to \omega/2$ in $X(e^{j\omega})$ and $e^{j\omega}X(e^{j\omega})$. That is, the DTFTs of the compressed sequences $g[n] = x[2n]$ and $h[n] = x[2n+1]$ are respectively

$$G(e^{j\omega}) = \frac{1}{2}\left(X(e^{j\omega/2}) + X(e^{j(\omega-2\pi)/2})\right) \tag{9.14a}$$

$$H(e^{j\omega}) = \frac{1}{2}\left(X(e^{j\omega/2})e^{j\omega/2} + X(e^{j(\omega-2\pi)/2})e^{j(\omega-2\pi)/2}\right). \tag{9.14b}$$

The sequence-expansion-by-2 shown in the right half of the block diagram in Figure 9.3 results in the frequency-compressed DTFTs $G_e(e^{j\omega}) = G(e^{j2\omega})$ and $H_e(e^{j\omega}) = H(e^{j2\omega})$, which, according to Figure 9.3, combine to form $X(e^{j\omega})$ through

$$\begin{aligned} X(e^{j\omega}) &= G_e(e^{j\omega}) + e^{-j\omega}H_e(e^{j\omega}) \\ &= G(e^{j2\omega}) + e^{-j\omega}H(e^{j2\omega}). \end{aligned} \tag{9.15}$$

Substituting Eqs. (9.14a) and (9.14b) into Eq. (9.15) will verify that the DTFT $X(e^{j\omega})$ of the N-point sequence $x[n]$ can be represented as in Eq. (9.15) in terms of the DTFTs of the $N/2$-point sequences $g[n] = x[2n]$ and $h[n] = x[2n+1]$. Therefore, the DFT $X[k]$ can likewise be represented in terms of the DFTs of $g[n]$ and $h[n]$.

Specifically, $X[k]$ corresponds to evaluating $X(e^{j\omega})$ at frequencies $\omega_k = 2\pi k/N$ with $k = 0, 1, \ldots, N-1$. Therefore, using Eq. (9.15) we obtain

$$X[k] = X(e^{j2\pi k/N}) = G(e^{j(2\pi k/N)2}) + e^{-j2\pi k/N}H(e^{j(2\pi k/N)2}). \tag{9.16}$$

From the definition of $g[n]$ and $G(e^{j\omega})$, it follows that

$$\begin{aligned}G(e^{j(2\pi k/N)2}) &= \sum_{n=0}^{N/2-1} x[2n]e^{-j(2\pi k/N)2n} \\ &= \sum_{n=0}^{N/2-1} x[2n]e^{-j(2\pi k/(N/2))n} \\ &= \sum_{n=0}^{N/2-1} x[2n]W_{N/2}^{kn}, \qquad (9.17a)\end{aligned}$$

and by a similar manipulation, it can be shown that

$$H(e^{j(2\pi k/N)2}) = \sum_{n=0}^{N/2-1} x[2n+1]W_{N/2}^{kn}. \qquad (9.17b)$$

Thus, from Eqs. (9.17a) and (9.17b) and Eq. (9.16), it follows that

$$X[k] = \sum_{n=0}^{N/2-1} x[2n]W_{N/2}^{kn} + W_N^k \sum_{n=0}^{N/2-1} x[2n+1]W_{N/2}^{kn} \qquad k = 0, 1, \ldots, N-1, \quad (9.18)$$

where the N-point DFT $X[k]$ is by definition

$$X[k] = \sum_{n=0}^{N-1} x[n]W_N^{nk}, \qquad k = 0, 1, \ldots, N-1. \qquad (9.19)$$

Likewise, by definition, the $(N/2)$-point DFTs of $g[n]$ and $h[n]$ are

$$G[k] = \sum_{n=0}^{N/2-1} x[2n]W_{N/2}^{nk}, \qquad k = 0, 1, \ldots, N/2-1 \qquad (9.20a)$$

$$H[k] = \sum_{n=0}^{N/2-1} x[2n+1]W_{N/2}^{nk}, \qquad k = 0, 1, \ldots, N/2-1. \qquad (9.20b)$$

Equation (9.18) shows that the N-point DFT $X[k]$ can be computed by evaluating the $(N/2)$-point DFTs $G[k]$ and $H[k]$ over $k = 0, 1, \ldots, N-1$ instead of $k = 0, 1, \ldots, N/2-1$ as we normally do for $(N/2)$-point DFTs. This is easily achieved even when $G[k]$ and $H[k]$ are computed only for $k = 0, 1, \ldots, N/2-1$, because the $(N/2)$-point transforms

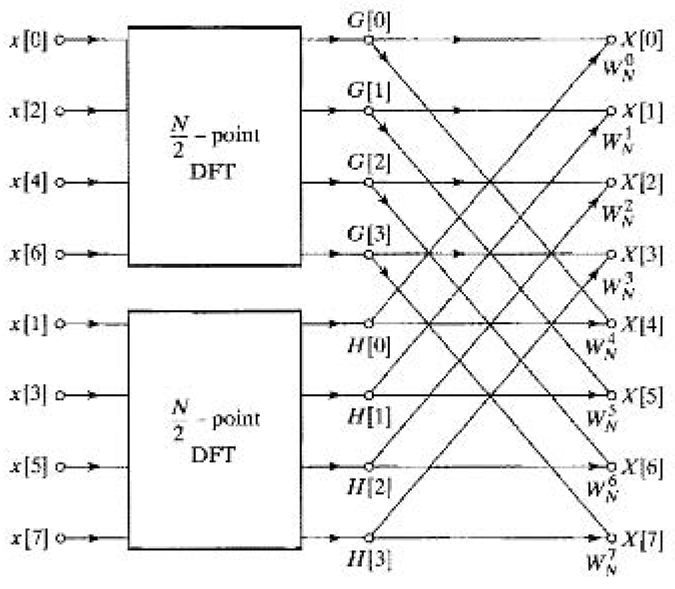


Figure 9.4 Flow graph of the decimation-in-time decomposition of an N-point DFT computation into two $(N/2)$-point DFT computations ($N = 8$).

are implicitly periodic with period $N/2$. With this observation, Eq. (9.18) can be rewritten as

$$X[k] = G[((k))_{N/2}] + W_N^k H[((k))_{N/2}] \qquad k = 0, 1, \ldots, N-1. \qquad (9.21)$$

The notation $((k))_{N/2}$ conveniently makes it explicit that even though $G[k]$ and $H[k]$ are only computed for $k = 0, 1, \ldots, N/2 - 1$, they are extended periodically (with no additional computation) by interpreting k modulo $N/2$.

After the two DFTs are computed, they are combined according to Eq. (9.21) to yield the N-point DFT $X[k]$. Figure 9.4 depicts this computation for $N = 8$. In this figure, we have used the signal flow graph conventions that were introduced in Chapter 6 for representing difference equations. That is, branches entering a node are summed to produce the node variable. When no coefficient is indicated, the branch transmittance is assumed to be unity. For other branches, the transmittance of a branch is an integer power of the complex number W_N.

In Figure 9.4, two 4-point DFTs are computed, with $G[k]$ designating the 4-point DFT of the even-numbered points and $H[k]$ designating the 4-point DFT of the odd-numbered points. According to Eq. (9.21), $X[0]$ is obtained by multiplying $H[0]$ by W_N^0 and adding the product to $G[0]$. The DFT value $X[1]$ is obtained by multiplying $H[1]$ by W_N^1 and adding that result to $G[1]$. Equation (9.21) states that, because of the implicit periodicity of $G[k]$ and $H[k]$, to compute $X[4]$, we should multiply $H[((4))_4]$ by W_N^4 and add the result to $G[((4))_4]$. Thus, $X[4]$ is obtained by multiplying $H[0]$ by W_N^4 and adding the result to $G[0]$. As shown in Figure 9.4, the values $X[5]$, $X[6]$, and $X[7]$ are obtained similarly.

With the computation restructured according to Eq. (9.21), we can compare the number of multiplications and additions required with those required for a direct computation of the DFT. Previously we saw that, for direct computation without exploiting symmetry, N^2 complex multiplications and additions were required.[6] By comparison, Eq. (9.21) requires the computation of two $(N/2)$-point DFTs, which in turn requires $2(N/2)^2$ complex multiplications and approximately $2(N/2)^2$ complex additions if we do the $(N/2)$-point DFTs by the direct method. Then the two $(N/2)$-point DFTs must be combined, requiring N complex multiplications, corresponding to multiplying the second sum by W_N^k, and N complex additions, corresponding to adding the product obtained to the first sum. Consequently, the computation of Eq. (9.21) for all values of k requires at most $N + 2(N/2)^2$ or $N + N^2/2$ complex multiplications and complex additions. It is easy to verify that for $N > 2$, the total $N + N^2/2$ will be less than N^2.

Equation (9.21) corresponds to breaking the original N-point computation into two $(N/2)$-point DFT computations. If $N/2$ is even, as it is when N is equal to a power of 2, then we can consider computing each of the $(N/2)$-point DFTs in Eq. (9.21) by breaking each of the sums in that equation into two $(N/4)$-point DFTs, which would then be combined to yield the $(N/2)$-point DFTs. Thus, $G[k]$ in Eq. (9.21) can be represented as

$$G[k] = \sum_{r=0}^{(N/2)-1} g[r]W_{N/2}^{rk} = \sum_{\ell=0}^{(N/4)-1} g[2\ell]W_{N/2}^{2\ell k} + \sum_{\ell=0}^{(N/4)-1} g[2\ell+1]W_{N/2}^{(2\ell+1)k}, \tag{9.22}$$

or

$$G[k] = \sum_{\ell=0}^{(N/4)-1} g[2\ell]W_{N/4}^{\ell k} + W_{N/2}^{k} \sum_{\ell=0}^{(N/4)-1} g[2\ell+1]W_{N/4}^{\ell k}. \tag{9.23}$$

Similarly, $H[k]$ can be represented as

$$H[k] = \sum_{\ell=0}^{(N/4)-1} h[2\ell]W_{N/4}^{\ell k} + W_{N/2}^{k} \sum_{\ell=0}^{(N/4)-1} h[2\ell+1]W_{N/4}^{\ell k}. \tag{9.24}$$

Consequently, the $(N/2)$-point DFT $G[k]$ can be obtained by combining the $(N/4)$-point DFTs of the sequences $g[2\ell]$ and $g[2\ell+1]$. Similarly, the $(N/2)$-point DFT $H[k]$ can be obtained by combining the $(N/4)$-point DFTs of the sequences $h[2\ell]$ and $h[2\ell+1]$. Thus, if the 4-point DFTs in Figure 9.4 are computed according to Eqs. (9.23) and (9.24), then that computation would be carried out as indicated in Figure 9.5. Inserting the computation of Figure 9.5 into the flow graph of Figure 9.4, we obtain the complete flow graph of Figure 9.6, where we have expressed the coefficients in terms of powers of W_N rather than powers of $W_{N/2}$, using the fact that $W_{N/2} = W_N^2$.

For the 8-point DFT that we have been using as an illustration, the computation has been reduced to a computation of 2-point DFTs. For example, the 2-point DFT of the sequence consisting of $x[0]$ and $x[4]$ is depicted in Figure 9.7. With the computation of Figure 9.7 inserted in the flow graph of Figure 9.6, we obtain the complete flow graph for computation of the 8-point DFT, as shown in Figure 9.9.

[6]For simplicity, we assume that N is large, so that $(N-1)$ can be approximated accurately by N.

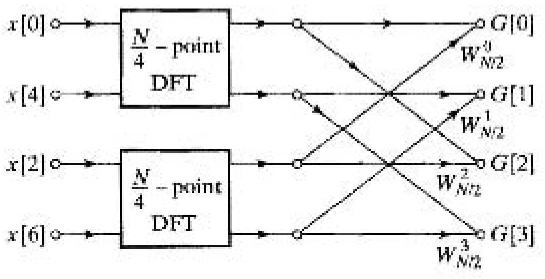


Figure 9.5 Flow graph of the decimation-in-time decomposition of an $(N/2)$-point DFT computation into two $(N/4)$-point DFT computations ($N = 8$).

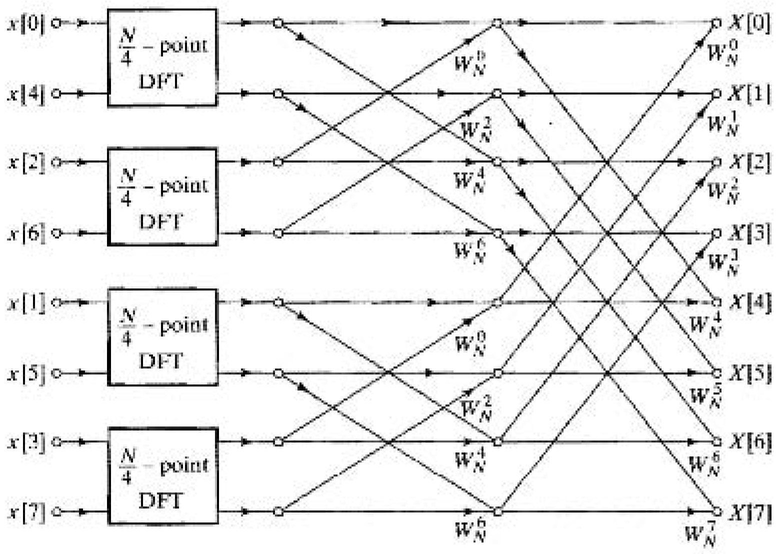


Figure 9.6 Result of substituting the structure of Figure 9.5 into Figure 9.4.

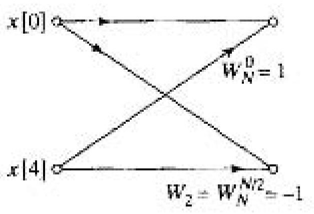


Figure 9.7 Flow graph of a 2-point DFT.

For the more general case, but with N still a power of 2, we would proceed by decomposing the $(N/4)$-point transforms in Eqs. (9.23) and (9.24) into $(N/8)$-point transforms and continue until we were left with only 2-point transforms. This requires $\nu = \log_2 N$ stages of computation. Previously, we found that in the original decomposition of an N-point transform into two $(N/2)$-point transforms, the number of complex multiplications and additions required was $N + 2(N/2)^2$. When the $(N/2)$-point transforms are decomposed into $(N/4)$-point transforms, the factor of $(N/2)^2$ is replaced by

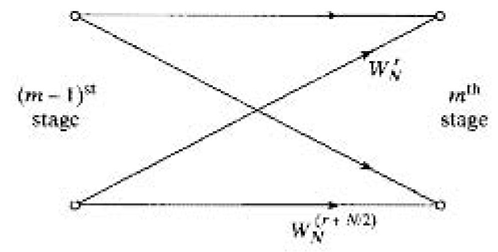


Figure 9.8 Flow graph of basic butterfly computation in Figure 9.9.

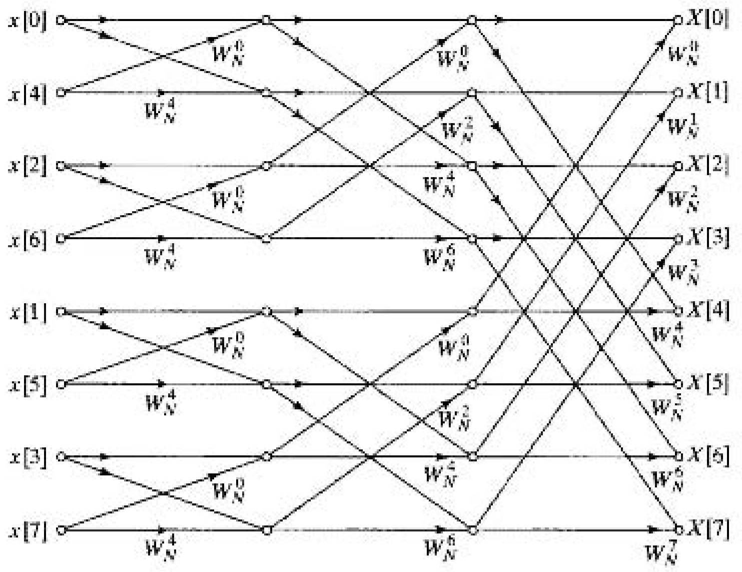


Figure 9.9 Flow graph of complete decimation-in-time decomposition of an 8-point DFT computation.

$N/2 + 2(N/4)^2$, so the overall computation then requires $N + N + 4(N/4)^2$ complex multiplications and additions. If $N = 2^\nu$, this can be done at most $\nu = \log_2 N$ times, so that after carrying out this decomposition as many times as possible, the number of complex multiplications and additions is equal to $N\nu = N \log_2 N$.

The flow graph of Figure 9.9 displays the operations explicitly. By counting branches with transmittances of the form W_N^r, we note that each stage has N complex multiplications and N complex additions. Since there are $\log_2 N$ stages, we have a total of $N \log_2 N$ complex multiplications and additions. This can be a substantial computational saving. For example, if $N = 2^{10} = 1024$, then $N^2 = 2^{20} = 1{,}048{,}576$, and $N \log_2 N = 10{,}240$, a reduction of more than two orders of magnitude!

The computation in the flow graph of Figure 9.9 can be reduced further by exploiting the symmetry and periodicity of the coefficients W_N^r. We first note that, in proceeding from one stage to the next in Figure 9.9, the basic computation is in the form of Figure 9.8, i.e., it involves obtaining a pair of values in one stage from a pair of values in the preceding stage, where the coefficients are always powers of W_N and the exponents are

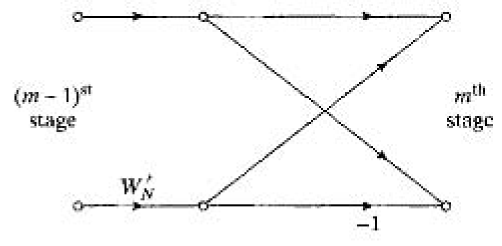


Figure 9.10 Flow graph of simplified butterfly computation requiring only one complex multiplication.

separated by $N/2$. Because of the shape of the flow graph, this elementary computation is called a *butterfly*. Since

$$W_N^{N/2} = e^{-j(2\pi/N)N/2} = e^{-j\pi} = -1, \tag{9.25}$$

the factor $W_N^{r+N/2}$ can be written as

$$W_N^{r+N/2} = W_N^{N/2}W_N^r = -W_N^r. \tag{9.26}$$

With this observation, the butterfly computation of Figure 9.8 can be simplified to the form shown in Figure 9.10, which requires one complex addition and one complex subtraction, but only one complex multiplication instead of two. Using the basic flow graph of Figure 9.10 as a replacement for butterflies of the form of Figure 9.8, we obtain from Figure 9.9 the flow graph of Figure 9.11. In particular, the number of complex multiplications has been reduced by a factor of 2 over the number in Figure 9.9.

Figure 9.11 shows $\log_2 N$ stages of computation each involving a set of $N/2$ 2-point DFT computations (butterflies). Between the sets of 2-point transforms are com-

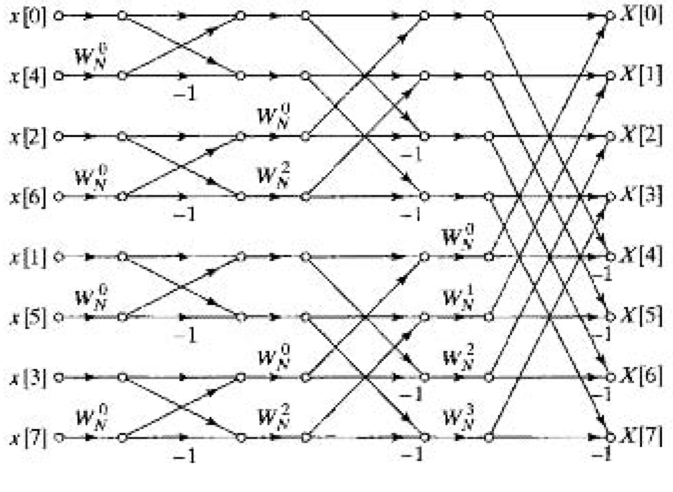


Figure 9.11 Flow graph of 8-point DFT using the butterfly computation of Figure 9.10.

plex multipliers of the form W_N^r. These complex multipliers have been called "twiddle factors" because they serve as adjustments in the process of converting the 2-point transforms into longer transforms.

9.2.1 Generalization and Programming the FFT

The flow graph of Figure 9.11, which describes an algorithm for computation of an 8-point discrete Fourier transform, is easily generalized to any $N = 2^\nu$, so it serves both as a proof that the computation requires on the order of $N \log N$ operations and as a graphical representation from which an implementation program could be written. While programs in high-level computer languages are widely available, it may be necessary in some cases to construct a program for a new machine architecture or to optimize a given program to take advantage of low-level features of a given machine architecture. A refined analysis of the diagram reveals many details that are important for programming or for designing special hardware for computing the DFT. We call attention to some of these details in Sections 9.2.2 and 9.2.3 for the decimation-in-time algorithms and in Sections 9.3.1 and 9.3.2 for the decimation-in-frequency algorithms. In Section 9.4 we discuss some additional practical considerations. While these sections are not essential for obtaining a basic understanding of FFT principles, they provide useful guidance for programming and system design.

9.2.2 In-Place Computations

The essential features of the flow graph of Figure 9.11 are the branches connecting the nodes and the transmittance of each of these branches. No matter how the nodes in the flow graph are rearranged, it will always represent the same computation, provided that the connections between the nodes and the transmittances of the connections are maintained. The particular form for the flow graph in Figure 9.11 arose out of deriving the algorithm by separating the original sequence into the even-numbered and odd-numbered points and then continuing to create smaller and smaller subsequences in the same way. An interesting by-product of this derivation is that this flow graph, in addition to describing an efficient procedure for computing the discrete Fourier transform, also suggests a useful way of storing the original data and storing the results of the computation in intermediate arrays.

To see this, it is useful to note that according to Figure 9.11, each stage of the computation takes a set of N complex numbers and transforms them into another set of N complex numbers through basic butterfly computations of the form of Figure 9.10. This process is repeated $\nu = \log_2 N$ times, resulting in the computation of the desired discrete Fourier transform. When implementing the computations depicted in Figure 9.11, we can imagine the use of two arrays of (complex) storage registers, one for the array being computed and one for the data being used in the computation. For example, in computing the first array in Figure 9.11, one set of storage registers would contain the input data and the second set would contain the computed results for the first stage. While the validity of Figure 9.11 is not tied to the order in which the input data are stored, we can order the set of complex numbers in the same order that they appear in the figure (from top to bottom). We denote the sequence of complex numbers re-

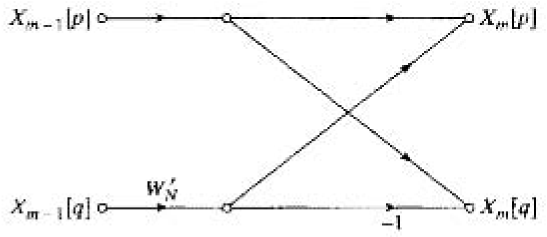


Figure 9.12 Flow graph of Eqs. (9.28).

sulting from the m^{th} stage of computation as $X_m[\ell]$, where $\ell = 0, 1, \ldots, N-1$, and $m = 1, 2, \ldots, \nu$. Furthermore, for convenience, we define the set of input samples as $X_0[\ell]$. We can think of $X_{m-1}[\ell]$ as the input array and $X_m[\ell]$ as the output array for the m^{th} stage of the computations. Thus, for the case of $N = 8$, as in Figure 9.11,

$$\begin{aligned} X_0[0] &= x[0], \\ X_0[1] &= x[4], \\ X_0[2] &= x[2], \\ X_0[3] &= x[6], \\ X_0[4] &= x[1], \\ X_0[5] &= x[5], \\ X_0[6] &= x[3], \\ X_0[7] &= x[7]. \end{aligned} \tag{9.27}$$

Using this notation, we can label the input and output of the butterfly computation in Figure 9.10 as indicated in Figure 9.12, with the associated equations

$$X_m[p] = X_{m-1}[p] + W_N^r X_{m-1}[q], \tag{9.28a}$$

$$X_m[q] = X_{m-1}[p] - W_N^r X_{m-1}[q]. \tag{9.28b}$$

In Eqs. (9.28), p, q, and r vary from stage to stage in a manner that is readily inferred from Figure 9.11 and from Eqs. (9.21), (9.23), and (9.24) and. It is clear from Figures 9.11 and 9.12 that only the complex numbers in locations p and q of the $(m-1)^{\text{st}}$ array are required to compute the elements p and q of the m^{th} array. Thus, only one complex array of N storage registers is physically necessary to implement the complete computation if $X_m[p]$ and $X_m[q]$ are stored in the same storage registers as $X_{m-1}[p]$ and $X_{m-1}[q]$, respectively. This kind of computation is commonly referred to as an *in-place* computation. The fact that the flow graph of Figure 9.11 (or Figure 9.9) represents an in-place computation is tied to the fact that we have associated nodes in the flow graph that are on the same horizontal line with the same storage location and the fact that the computation between two arrays consists of a butterfly computation in which the input nodes and the output nodes are horizontally adjacent.

In order that the computation may be done in place as just discussed, the input sequence must be stored (or at least accessed) in a nonsequential order, as shown in the flow graph of Figure 9.11. In fact, the order in which the input data are stored and accessed is referred to as *bit-reversed* order. To see what is meant by this terminology, we note that for the 8-point flow graph that we have been discussing, three binary digits

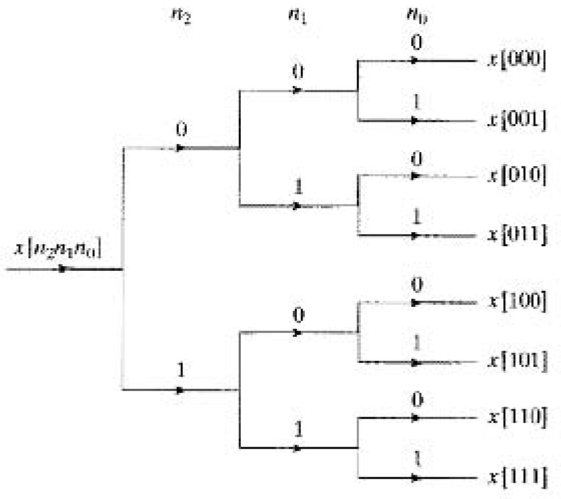


Figure 9.13 Tree diagram depicting normal-order sorting.

are required to index through the data. Writing the indices in Eqs. (9.27) in binary form, we obtain the following set of equations:

$$\begin{aligned}
X_0[000] &= x[000],\\
X_0[001] &= x[100],\\
X_0[010] &= x[010],\\
X_0[011] &= x[110],\\
X_0[100] &= x[001],\\
X_0[101] &= x[101],\\
X_0[110] &= x[011],\\
X_0[111] &= x[111].
\end{aligned} \tag{9.29}$$

If (n_2, n_1, n_0) is the binary representation of the index of the sequence $x[n]$, then the sequence value $x[n_2, n_1, n_0]$ is stored in the array position $X_0[n_0, n_1, n_2]$. That is, in determining the position of $x[n_2, n_1, n_0]$ in the input array, we must reverse the order of the bits of the index n.

Consider the process depicted in Figure 9.13 for sorting a data sequence in normal order by successive examination of the bits representing the data index. If the most significant bit of the data index is zero, $x[n]$ belongs in the top half of the sorted array; otherwise it belongs in the bottom half. Next, the top half and bottom half subsequences can be sorted by examining the second most significant bit, and so on.

To see why bit-reversed order is necessary for in-place computation, recall the process that resulted in Figure 9.9 and Figure 9.11. The sequence $x[n]$ was first divided into the even-numbered samples, with the even-numbered samples occurring in the top half of Figure 9.4 and the odd-numbered samples occurring in the bottom half. Such a separation of the data can be carried out by examining the least significant bit $[n_0]$ in the index n. If the least significant bit is 0, the sequence value corresponds to an even-numbered sample and therefore will appear in the top half of the array $X_0[\ell]$, and if the least significant bit is 1, the sequence value corresponds to an odd-numbered

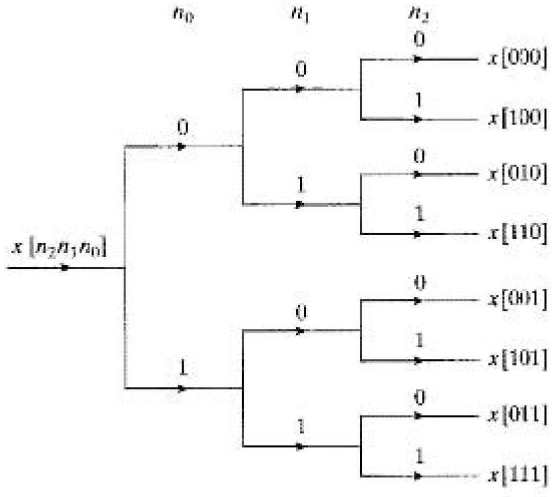


Figure 9.14 Tree diagram depicting bit-reversed sorting.

sample and consequently will appear in the bottom half of the array. Next, the even- and odd-indexed subsequences are sorted into their even- and odd-indexed parts, and this can be done by examining the second least significant bit in the index. Considering first the even-indexed subsequence, if the second least significant bit is 0, the sequence value is an even-numbered term in the subsequence, and if the second least significant bit is 1, then the sequence value has an odd-numbered index in this subsequence. The same process is carried out for the subsequence formed from the original odd-indexed sequence values. This process is repeated until N subsequences of length 1 are obtained. This sorting into even- and odd-indexed subsequences is depicted by the tree diagram of Figure 9.14.

The tree diagrams of Figures 9.13 and 9.14 are identical, except that for normal sorting, we examine the bits representing the index from left to right, whereas for the sorting leading naturally to Figure 9.9 or 9.11, we examine the bits in reverse order, right to left, resulting in bit-reversed sorting. Thus, the necessity for bit-reversed ordering of the sequence $x[n]$ results from the manner in which the DFT computation is decomposed into successively smaller DFT computations in arriving at Figures 9.9 and 9.11.

9.2.3 Alternative Forms

Although it is reasonable to store the results of each stage of the computation in the order in which the nodes appear in Figure 9.11, it is certainly not necessary to do so. No matter how the nodes of Figure 9.11 are rearranged, the result will always be a valid computation of the discrete Fourier transform of $x[n]$, as long as the branch transmittances are unchanged. Only the order in which data are accessed and stored will change. If we associate the nodes with indexing of an array of complex storage locations, it is clear from our previous discussion that a flow graph corresponding to an in-place computation results only if the rearrangement of nodes is such that the input and output nodes for each butterfly computation are horizontally adjacent. Otherwise two complex storage arrays will be required. Figure 9.11, is, of course, such an arrangement. Another is depicted in Figure 9.15. In this case, the input sequence is in normal order and the sequence of DFT values is in bit-reversed order. Figure 9.15 can be obtained from Figure 9.11 as follows: All the nodes that are horizontally adjacent to $x[4]$ in Figure 9.11 are interchanged with all the nodes horizontally adjacent to $x[1]$. Similarly, all the nodes that are horizontally adjacent to $x[6]$ in Figure 9.11 are interchanged with those that are horizontally adjacent to $x[3]$. The nodes horizontally adjacent to $x[0]$, $x[2]$, $x[5]$, and $x[7]$ are not disturbed. The resulting flow graph in Figure 9.15 corresponds to the form of the decimation-in-time algorithm originally given by Cooley and Tukey (1965).

The only difference between Figures 9.11 and 9.15 is in the ordering of the nodes. This implies that Figures 9.11 and 9.15 represent two different programs for carrying out the computations. The branch transmittances (powers of W_N) remain the same, and therefore the intermediate results will be exactly the same—they will be computed in a different order within each stage. There are, of course, a large variety of possible orderings. However, most do not make much sense from a computational viewpoint. As one example, suppose that the nodes are ordered such that the input and output both appear in normal order. A flow graph of this type is shown in Figure 9.16. In this case, however, the computation cannot be carried out in place because the butterfly structure does not continue past the first stage. Thus, two complex arrays of length N would be required to perform the computation depicted in Figure 9.16.

In realizing the computations depicted by Figures 9.11, 9.15, and 9.16, it is clearly necessary to access elements of intermediate arrays in non-sequential order. Thus, for greater computational speed, the complex numbers must be stored in random-access memory.[7] For example, in the computation of the first array in Figure 9.11 from the input array, the inputs to each butterfly computation are adjacent node variables and are thought of as being stored in adjacent storage locations. In the computation of the second intermediate array from the first, the inputs to a butterfly are separated by two storage locations; and in the computation of the third array from the second, the inputs to a butterfly computation are separated by four storage locations. If $N > 8$, the separation between butterfly inputs is 8 for the fourth stage, 16 for the fifth stage, and so on. The separation in the last (ν^{th}) stage is $N/2$.

In Figure 9.15 the situation is similar in that, to compute the first array from the input data we use data separated by 4, to compute the second array from the first array we use input data separated by 2, and then finally, to compute the last array we use adjacent data. It is straightforward to imagine simple algorithms for modifying index registers to access the data in the flow graph of either Figure 9.11 or Figure 9.15 if the data are stored in random-access memory. However, in the flow graph of Figure 9.16, the data are accessed non-sequentially, the computation is not in place, and a scheme for indexing the data is considerably more complicated than in either of the two previous cases. Even given the availability of large amounts of random-access memory, the overhead for index computations could easily nullify much of the computational advantage that is implied by eliminating multiplications and additions. Consequently, this structure has no apparent advantages.

[7]When the Cooley–Tukey algorithms first appeared in 1965, digital memory was expensive and of limited size. The size and availability of random access memory is no longer an issue except for exceedingly large values of N.

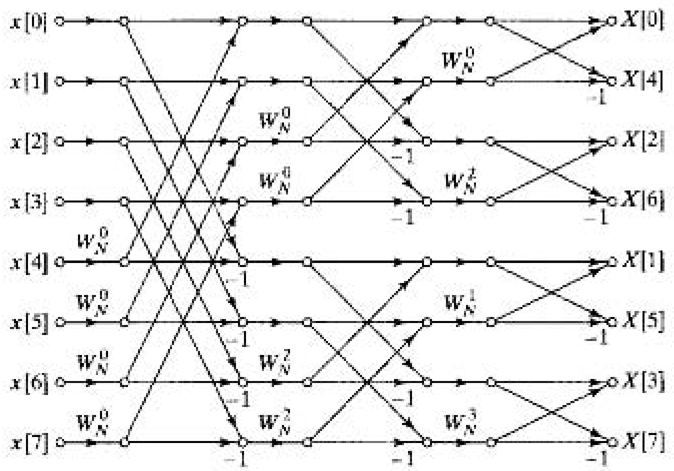


Figure 9.15 Rearrangement of Figure 9.11 with input in normal order and output in bit-reversed order.

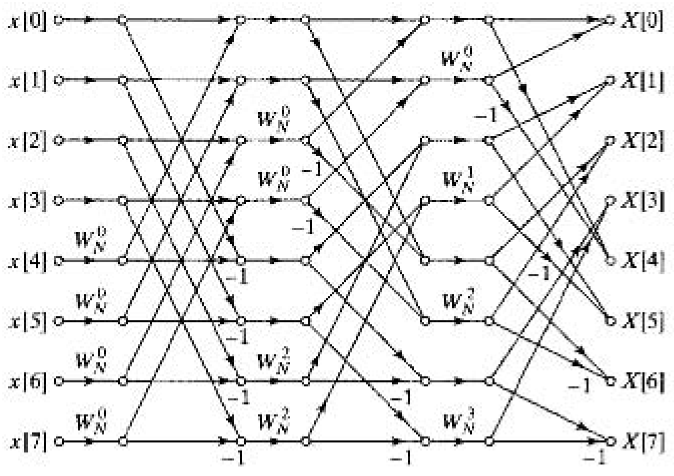


Figure 9.16 Rearrangement of Figure 9.11 with both input and output in normal order.

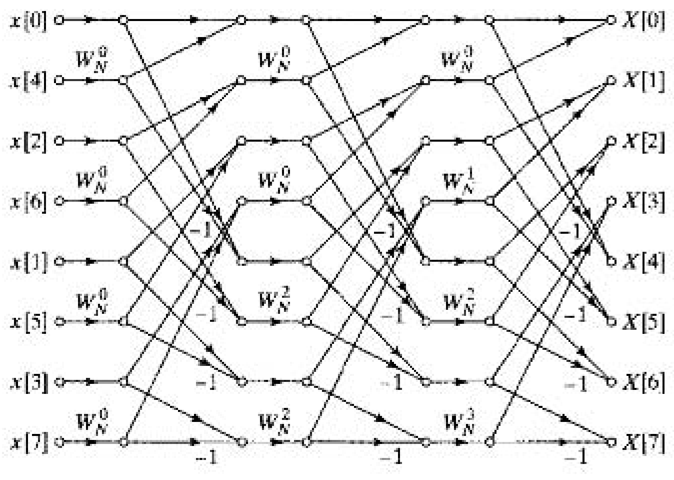


Figure 9.17 Rearrangement of Figure 9.11 having the same geometry for each stage, thereby simplifying data access.

Some forms have advantages even if they do not allow in-place computation. A rearrangement of the flow graph in Figure 9.11 that is particularly useful when an adequate amount of random-access memory is not available is shown in Figure 9.17. This flow graph represents the decimation-in-time algorithm originally given by Singleton (1969). Note first that in this flow graph the input is in bit-reversed order and the output is in normal order. The important feature of the flow graph is that the geometry is identical for each stage; only the branch transmittances change from stage to stage. This makes it possible to access data sequentially. Suppose, for example that we have four separate mass-storage files, and suppose that the first half of the input sequence (in bit-reversed order) is stored in one file and the second half is stored in a second file. Then the sequence can be accessed sequentially in files 1 and 2 and the results written sequentially on files 3 and 4, with the first half of the new array being written to file 3 and the second half to file 4. Then at the next stage of computation, files 3 and 4 are the input, and the output is written to files 1 and 2. This is repeated for each of the ν stages.

Such an algorithm could be useful in computing the DFT of extremely long sequences. This could mean values of N on the order of hundreds of millions since random-access memories of giga-byte size are routinely available. Perhaps a more interesting feature of the diagram in Figure 9.17 is that the indexing is very simple and it is the same from stage-to-stage. With two banks of random-access memory, this algorithm would have very simple index calculations.

9.3 DECIMATION-IN-FREQUENCY FFT ALGORITHMS

The decimation-in-time FFT algorithms are based on structuring the DFT computation by forming smaller and smaller subsequences of the input sequence $x[n]$. Alternatively, we can consider dividing the DFT sequence $X[k]$ into smaller and smaller subsequences in the same manner. FFT algorithms based on this procedure are commonly called *decimation-in-frequency* algorithms.

To develop this class of FFT algorithms, we again restrict the discussion to N a power of 2 and consider computing separately the $N/2$ even-numbered frequency samples and the $N/2$ odd-numbered frequency samples. We have depicted this in the block diagram representation in Figure 9.18 where $X_0[k] = X[2k]$ and $X_1[k] = X[2k+1]$. In shifting left by 1 DFT sample so that the compressor selects the odd-indexed samples, it is important to remember that the DFT $X[k]$ is implicitly periodic with period N. This is denoted "Circular Left Shift 1" (and correspondingly "Circular Right Shift 1") in Figure 9.18. Observe that this diagram has a similar structure to Figure 9.3, where the same operations were applied to the time sequence $x[n]$ instead of the DFT $X[k]$. In this case, Figure 9.18 directly depicts the fact that the N-point transform $X[k]$ can be obtained by interleaving its even-indexed and odd-indexed samples after expansion by a factor of 2.

Figure 9.18 is a correct representation of $X[k]$, but in order to use it as the basis for computing $X[k]$, we first show that $X[2k]$ and $X[2k+1]$ can be computed from the time-domain sequence $x[n]$. In Section 8.4 we saw that the DFT is related to the DTFT by sampling at frequencies $2\pi k/N$ with the result that the corresponding time-

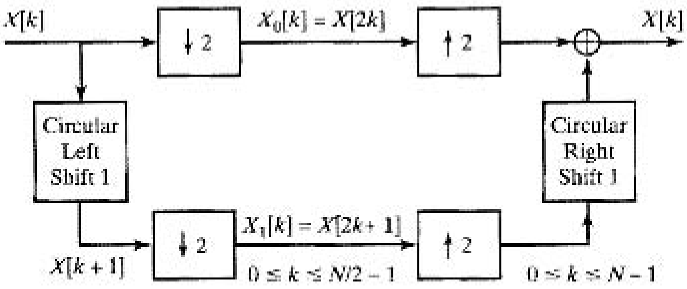


Figure 9.18 Illustration of the basic principle of decimation-in-frequency.

domain operation is time-aliasing with repetition length (period) N. As discussed in Section 8.4, if N is greater than or equal to the length of the sequence $x[n]$, the inverse DFT yields the original sequence over $0 \le n \le N-1$ because the N-point copies of $x[n]$ do not overlap when time-aliased with repetition offset N. However, in Figure 9.18, the DFT is compressed by 2, which is equivalent to sampling the DTFT $X(e^{j\omega})$ at frequencies $2\pi k/(N/2)$. Thus, the implicit periodic time-domain signal represented by $X_0[k] = X[2k]$ is

$$\tilde{x}_0[n] = \sum_{m=-\infty}^{\infty} x[n+mN/2] \qquad -\infty < n < \infty. \tag{9.30}$$

Since $x[n]$ has length N, only two of the shifted copies of $x[n]$ overlap in the interval $0 \le n \le N/2-1$, so the corresponding finite-length sequence $x_0[n]$ is

$$x_0[n] = x[n] + x[n+N/2] \qquad 0 \le n \le N/2-1. \tag{9.31a}$$

To obtain the comparable result for the odd-indexed DFT samples, recall that the circularly shifted DFT $X[k+1]$ corresponds to $W_N^n x[n]$ (see Property 6 of Table 8.2). Therefore the $N/2$-point sequence $x_1[n]$ corresponding to $X_1[k] = X[2k+1]$ is

$$\begin{aligned} x_1[n] &= x[n]W_N^n + x[n+N/2]W_N^{n+N/2} \\ &= (x[n] - x[n+N/2])W_N^n \qquad 0 \le n \le N/2-1, \end{aligned} \tag{9.31b}$$

since $W_N^{N/2} = -1$.

From Eqs. (9.31a) and (9.31b), it follows that

$$X_0[k] = \sum_{n=0}^{N/2-1} (x[n] + x[n+N/2])W_{N/2}^{kn} \tag{9.32a}$$

$$X_1[k] = \sum_{n=0}^{N/2-1} [(x[n] - x[n+N/2])W_N^n]W_{N/2}^{kn} \tag{9.32b}$$

$$k = 0, 1, \ldots, N/2-1.$$

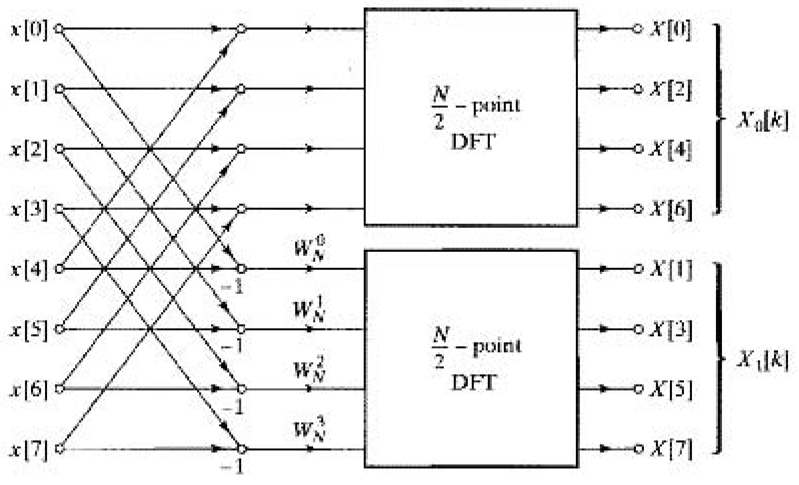


Figure 9.19 Flow graph of decimation-in-frequency decomposition of an N-point DFT computation into two $(N/2)$-point DFT computations ($N = 8$).

Equation (9.32a) is the $(N/2)$-point DFT of the sequence $x_0[n]$ obtained by adding the second half of the input sequence to the first half. Equation (9.32b) is the $(N/2)$-point DFT of the sequence $x_1[n]$ obtained by subtracting the second half of the input sequence from the first half and multiplying the resulting sequence by W_N^n.

Thus, using Eqs. (9.32a) and (9.32b), the even-numbered and odd-numbered output points of $X[k]$ can be computed since $X[2k] = X_0[k]$ and $X[2k+1] = X_1[k]$, respectively. The procedure suggested by Eqs. (9.32a) and (9.32b) is illustrated for the case of an 8-point DFT in Figure 9.19.

Proceeding in a manner similar to that followed in deriving the decimation-in-time algorithm, we note that for N a power of 2, $N/2$ is divisible by 2 so the $(N/2)$-point DFTs can be computed by computing the even-numbered and odd numbered output points for those DFTs separately. As in the case of the procedure leading to Eqs. (9.32a) and (9.32b), this is accomplished by combining the first half and the last half of the input points for each of the $(N/2)$-point DFTs and then computing $(N/4)$-point DFTs. The flow graph resulting from taking this step for the 8-point example is shown in Figure 9.20. For the 8-point example, the computation has now been reduced to the computation of 2-point DFTs, which are implemented by adding and subtracting the input points, as discussed previously. Thus, the 2-point DFTs in Figure 9.20 can be replaced by the computation shown in Figure 9.21, so the computation of the 8-point DFT can be accomplished by the algorithm depicted in Figure 9.22. We again see $\log_2 N$ stages of 2-point transforms coupled together through twiddle factors that in this case occur at the output of the 2-point transforms.

By counting the arithmetic operations in Figure 9.22 and generalizing to $N = 2^\nu$, we see that the computation of Figure 9.22 requires $(N/2)\log_2 N$ complex multiplications and $N\log_2 N$ complex additions. Thus, the total number of computations is the same for the decimation-in-frequency and the decimation-in-time algorithms.

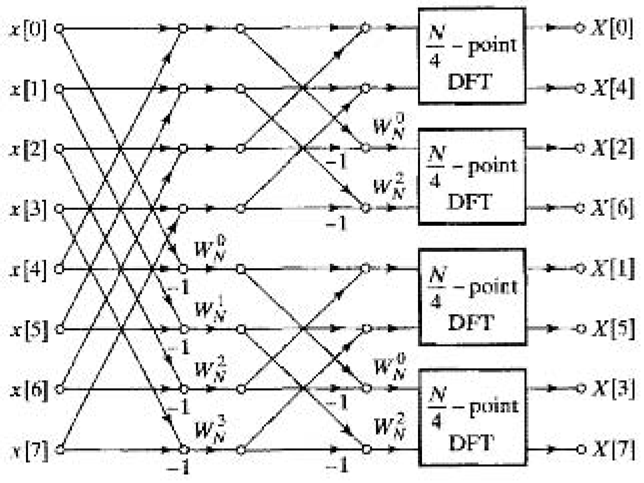


Figure 9.20 Flow graph of decimation-in-frequency decomposition of an 8-point DFT into four 2-point DFT computations.

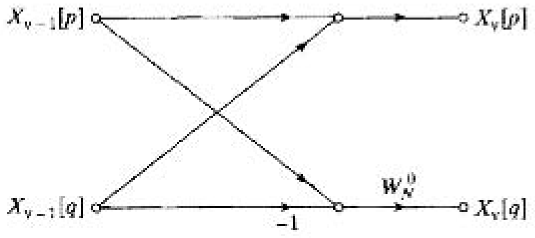


Figure 9.21 Flow graph of a typical 2-point DFT as required in the last stage of decimation-in-frequency decomposition.

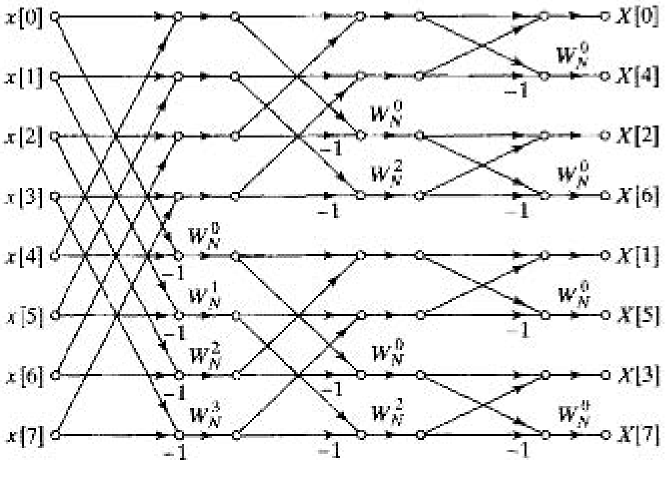


Figure 9.22 Flow graph of complete decimation-in-frequency decomposition of an 8-point DFT computation.

9.3.1 In-Place Computation

The flow graph in Figure 9.22 depicts one FFT algorithm based on decimation in frequency. We can observe a number of similarities and also a number of differences in comparing this graph with the flow graphs derived on the basis of decimation in time. As with decimation in time, of course, the flow graph of Figure 9.22 corresponds to a computation of the discrete Fourier transform, regardless of how the graph is drawn, as long as the same nodes are connected to each other with the proper branch transmittances. In other words, the flow graph of Figure 9.22 is not based on any assumption about the order in which the input sequence values are stored. However, as was done with the decimation-in-time algorithms, we can interpret successive vertical nodes in the flow graph of Figure 9.22 as corresponding to successive storage registers in a digital memory. In this case, the flow graph in Figure 9.22 begins with the input sequence in normal order and provides the output DFT in bit-reversed order. The basic computation again has the form of a butterfly computation, although the butterfly is different from that arising in the decimation-in-time algorithms. However, because of the butterfly nature of the computation, the flow graph of Figure 9.22 can be interpreted as an in-place computation of the discrete Fourier transform.

9.3.2 Alternative Forms

A variety of alternative forms for the decimation-in-frequency algorithm can be obtained by transposing the decimation-in-time forms developed in Section 9.2.3. If we denote the sequence of complex numbers resulting from the m^{th} stage of the computation as $X_m[\ell]$, where $\ell = 0, 1, \dots, N-1$, and $m = 1, 2, \dots, \nu$, then the basic butterfly computation shown in Figure 9.23 has the form

$$X_m[p] = X_{m-1}[p] + X_{m-1}[q], \tag{9.33a}$$

$$X_m[q] = (X_{m-1}[p] - X_{m-1}[q])W_N^r. \tag{9.33b}$$

Comparing Figures 9.12 and 9.23 or Eqs. (9.28) and (9.33), it appears that the butterfly computations are different for the two classes of FFT algorithms. However, the two butterfly flow graphs are, in the terminology of Chapter 6, transposes of one another. That is, if we reverse the direction of arrows and redefine the input and output nodes in Figure 9.12, we obtain Figure 9.23 and vice-versa. Since the FFT flow graphs consist of connected sets of butterflies, it is not surprising, therefore, that we also note

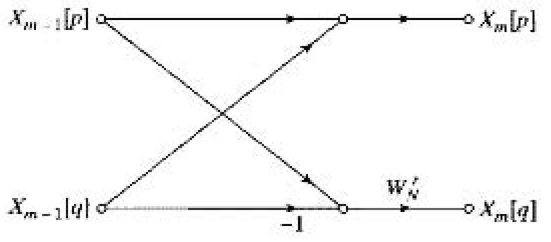


Figure 9.23 Flow graph of a typical butterfly computation required in Figure 9.22.

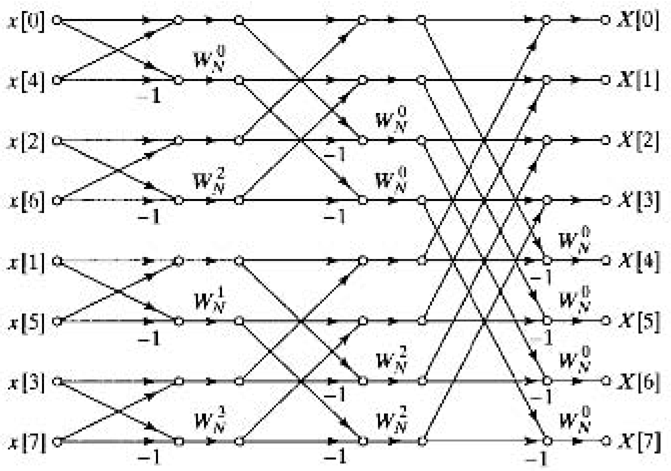


Figure 9.24 Flow graph of a decimation-in-frequency DFT algorithm obtained from Figure 9.22. Input in bit-reversed order and output in normal order. (Transpose of Figure 9.15.)

a resemblance between the FFT flow graphs of Figures 9.11 and 9.22. Specifically, Figure 9.22 can be obtained from Figure 9.11 by reversing the direction of signal flow and interchanging the input and output. That is, Figure 9.22 is the transpose of the flow graph in Figure 9.11. In Chapter 6 we stated a transposition theorem that applies only to single-input/single-output flow graphs. When viewed as flow graphs, however, FFT algorithms are multi-input/multi-output systems, which require a more general form of the transposition theorem. (See Claasen and Mecklenbräuker, 1978.) Nevertheless, it is intuitively clear that the input–output characteristics of the flow graphs in Figures 9.11 and 9.22 are the same based simply on the above observation that the butterflies are transposes of each other. This can be shown more formally by noting that the butterfly equations in Eqs. (9.33) can be solved backward, starting with the output array. (Problem 9.31 outlines a proof of this result.) More generally, it is true that for each decimation-in-time FFT algorithm, there exists a decimation-in-frequency FFT algorithm that corresponds to interchanging the input and output and reversing the direction of all the arrows in the flow graph.

This result implies that all the flow graphs of Section 9.2 have counterparts in the class of decimation-in-frequency algorithms. This, of course, also corresponds to the fact that, as before, it is possible to rearrange the nodes of a decimation-in-frequency flow graph without altering the final result.

Applying the transposition procedure to Figure 9.15 leads to Figure 9.24. In this flow graph, the output is in normal order and the input is in bit-reversed order. The transpose of the flow graph of Figure 9.16 would lead to a flow graph with both the input and output in normal order. An algorithm base on the resulting flow graph would suffer from the same limitations as for Figure 9.16.

The transpose of Figure 9.17 is shown in Figure 9.25. Each stage of Figure 9.25 has the same geometry, a property that simplifies data access, as discussed before.

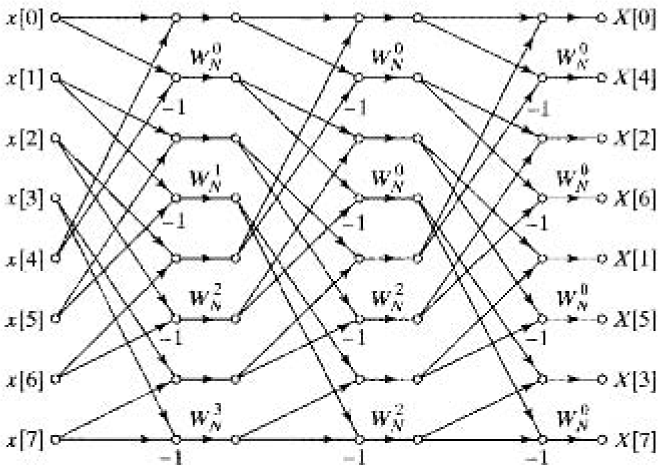


Figure 9.25 Rearrangement of Figure 9.22 having the same geometry for each stage, thereby simplifying data access. (Transpose of Figure 9.17.)

9.4 PRACTICAL CONSIDERATIONS

In Sections 9.2 and 9.3, we discussed the basic principles of efficient computation of the DFT when N is an integer power of 2. In these discussions, we favored the use of signal flow graph representations rather than explicitly writing out in detail the equations that such flow graphs represent. Of necessity, we have shown flow graphs for specific values of N. However, by considering a flow graph such as that in Figure 9.11, for a specific value of N, it is possible to see how to structure a general computational algorithm that would apply to any $N = 2^\nu$. While the discussion in Sections 9.2 and 9.3 is completely adequate for a basic understanding of the FFT principles, the material of this section is intended to provide useful guidance for programming and system design.

Although it is true that the flow graphs of the previous sections capture the essence of the FFT algorithms that they depict, a variety of details must be considered in the implementation of a given algorithm. In this section, we briefly suggest some of these. Specifically, in Section 9.4.1 we discuss issues associated with accessing and storing data in the intermediate arrays of the FFT. In Section 9.4.2 we discuss issues associated with computing or accessing the branch coefficients in the flow graph. Our emphasis is on algorithms for N a power of 2, but much of the discussion applies to the general case as well. For purposes of illustration, we focus primarily on the decimation-in-time algorithm of Figure 9.11.

9.4.1 Indexing

In the algorithm depicted in Figure 9.11, the input must be in bit-reversed order so that the computation can be performed in place. The resulting DFT is then in normal order. Generally, sequences do not originate in bit-reversed order, so the first step in the implementation of Figure 9.11 is to sort the input sequence into bit-reversed order. As can be seen from that figure and Eqs. (9.27) and (9.29), bit-reversed sorting can be done in place, since samples are only pairwise interchanged; i.e., a sample at a given index is interchanged with the sample in the location specified by the bit-reversed index. This is conveniently done in place by using two counters, one in normal order and the other in bit-reversed order. The data in the two positions specified by the two counters are simply interchanged. Once the input is in bit-reversed order, we can proceed with the first stage of computation. In this case, the inputs to the butterflies are adjacent elements of the array $X_0[\cdot]$. In the second stage, the inputs to the butterflies are separated by 2. In the m^{th} stage, the butterfly inputs are separated by 2^{m-1}. The coefficients are powers of $W_N^{(N/2^m)}$ in the m^{th} stage and are required in normal order if computation of butterflies begins at the top of the flow graph of Figure 9.11. The preceding statements define the manner in which data must be accessed at a given stage, which, of course, depends on the flow graph that is implemented. For example, in the m^{th} stage of Figure 9.15, the butterfly spacing is $2^{\nu-m}$, and in this case the coefficients are required in bit-reversed order. The input is in normal order; however, the output is in bit-reversed order, so it generally would be necessary to sort the output into normal order by using a normal-order counter and a bit-reversed counter, as discussed previously.

In general, if we consider all the flow graphs in Sections 9.2 and 9.3, we see that each algorithm has its own characteristic indexing issues. The choice of a particular algorithm depends on a number of factors. The algorithms utilizing an in-place computation have the advantage of making efficient use of memory. Two disadvantages, however, are that the kind of memory required is random-access rather than sequential memory and that either the input sequence or the output DFT sequence is in bit-reversed order. Furthermore, depending on whether a decimation-in-time or a decimation-in-frequency algorithm is chosen and whether the inputs or the outputs are in bit-reversed order, the coefficients are required to be accessed in either normal order or bit-reversed order. If non-random-access sequential memory is used, some fast Fourier transform algorithms utilize sequential memory, as we have shown, but either the inputs or the outputs must be in bit-reversed order. While the flow graph for the algorithm can be arranged so that the inputs, the outputs, and the coefficients are in normal order, the indexing structure required to implement these algorithms is complicated, and twice as much random access memory is required. Consequently, the use of such algorithms does not appear to be advantageous.

The in-place FFT algorithms of Figures 9.11, 9.15, 9.22, and 9.24 are among the most commonly used. If a sequence is to be transformed only once, then bit-reversed sorting must be implemented on either the input or the output. However, in some situations a sequence is transformed, the result is modified in some way, and then the inverse DFT is computed. For example, in implementing FIR digital filters by block convolution using the discrete Fourier transform, the DFT of a section of the input sequence is multiplied by the DFT of the filter impulse response, and the result is inverse transformed to obtain a segment of the output of the filter. Similarly, in computing an autocorrelation function or cross-correlation function using the discrete Fourier transform, a sequence will be transformed, the DFTs will be multiplied, and then the resulting product will be inverse transformed. When two transforms are cascaded in this way, it is possible, by appropriate choice of the FFT algorithms, to avoid the need for bit reversal. For example, in implementing an FIR digital filter using the DFT, we can choose an algorithm for the direct transform that utilizes the data in normal order and provides a DFT in bit-reversed order. Either the flow graph corresponding to Figure 9.15, based on decimation in time, or that of Figure 9.22, based on decimation in frequency, could be used in this way. The difference between these two forms is that the decimation-in-time form requires the coefficients in bit-reversed order, whereas the decimation-in-frequency form requires the coefficients in normal order.

Note that Figure 9.11 utilizes coefficients in normal order, whereas Figure 9.24 requires the coefficients in bit-reversed order. If the decimation-in-time form of the algorithm is chosen for the direct transform, then the decimation-in-frequency form of the algorithm should be chosen for the inverse transform, requiring coefficients in bit-reversed order. Likewise, the decimation-in-frequency algorithm for the direct transform should be paired with the decimation-in-time algorithm for the inverse transform, which would then utilize normally ordered coefficients.

9.4.2 Coefficients

We have observed that the coefficients W_N^r (twiddle factors) may be required in either bit-reversed order or in normal order. In either case we must store a table sufficient to look up all required values, or we must compute the values as needed. The first alternative has the advantage of speed, but of course requires extra storage. We observe from the flow graphs that we require W_N^r for $r = 0, 1, \ldots, (N/2) - 1$. Thus, we require $N/2$ complex storage registers for a complete table of values of W_N^r.[8] In the case of algorithms in which the coefficients are required in bit-reversed order, we can simply store the table in bit-reversed order.

The computation of the coefficients as they are needed saves storage, but is less efficient than storing a lookup table. If the coefficients are to be computed, it is generally most efficient to use a recursion formula. At any given stage, the required coefficients are all powers of a complex number of the form W_N^q, where q depends on the algorithm and the stage. Thus, if the coefficients are required in normal order, we can use the recursion formula

$$W_N^{q\ell} = W_N^q \cdot W_N^{q(\ell-1)} \tag{9.34}$$

to obtain the ℓ^{th} coefficient from the $(\ell-1)^{\text{st}}$ coefficient. Clearly, algorithms that require coefficients in bit-reversed order are not well suited to this approach. It should be noted that Eq. (9.34) is essentially the coupled-form oscillator of Problem 6.21. When using finite-precision arithmetic, errors can build up in the iteration of this difference equation. Therefore, it is generally necessary to reset the value at prescribed points (e.g., $W_N^{N/4} = -j$) so that errors do not become unacceptable.

9.5 MORE GENERAL FFT ALGORITHMS

The power-of-two algorithms discussed in detail in Sections 9.2 and 9.3 are straightforward, highly efficient and easy to program. However, there are many applications where efficient algorithms for other values of N are very useful.

[8]This number can be reduced using symmetry at the cost of greater complexity in accessing desired values.

9.5.1 Algorithms for Composite Values of N

Although the special case of N a power of 2 leads to algorithms that have a particularly simple structure, this is not the only restriction on N that can lead to reduced computation for the DFT. The same principles that were applied in the power-of-two decimation-in-time and decimation-in-frequency algorithms can be employed when N is a composite integer, i.e., the product of two or more integer factors. For example, if $N = N_1N_2$, it is possible to express the N-point DFT as either a combination of N_1 N_2-point DFTs or as a combination of N_2 N_1-point DFTs, and thereby obtain reductions in the number of computations. To see this, the indices n and k are represented as follows:

$$n = N_2n_1 + n_2 \qquad \begin{cases} n_1 = 0, 1, \ldots, N_1 - 1 \\ n_2 = 0, 1, \ldots, N_2 - 1 \end{cases} \tag{9.35a}$$

$$k = k_1 + N_1k_2 \qquad \begin{cases} k_1 = 0, 1, \ldots, N_1 - 1 \\ k_2 = 0, 1, \ldots, N_2 - 1. \end{cases} \tag{9.35b}$$

Since $N = N_1N_2$, these index decompositions ensure that n and k range over all the values $0, 1, \ldots, N - 1$. Substituting these representations of n and k into the definition of the DFT leads after a few manipulations to

$$\begin{aligned} X[k] &= X[k_1 + N_1k_2] \\ &= \sum_{n_2=0}^{N_2-1}\left[\left(\sum_{n_1=0}^{N_1-1} x[N_2n_1 + n_2]W_{N_1}^{k_1n_1}\right)W_N^{k_1n_2}\right]W_{N_2}^{k_2n_2}, \end{aligned} \tag{9.36}$$

where $k_1 = 0, 1, \ldots, N_1 - 1$ and $k_2 = 0, 1, \ldots, N_2 - 1$. The part of Eq. (9.36) inside the parentheses represents N_2 N_1-point DFTs, while the outer sum corresponds to N_1 N_2-point DFTs of the outputs of the first set of transforms occurring after modification by the twiddle factors $W_N^{k_1n_2}$.

If $N_1 = 2$ and $N_2 = N/2$, Eq. (9.36) reduces to the first stage decomposition of the decimation-in-frequency power-of-two algorithm depicted in Figure 9.19 of Section 9.3, which consists of $N/2$ 2-point transforms followed by two $N/2$-point transforms. Conversely, if $N_1 = N/2$ and $N_2 = 2$, Eq. (9.36) reduces to the first stage decomposition of the decimation-in-time power-of-two algorithm depicted in Figure 9.4 Section 9.2, which consists of two $N/2$-point transforms followed by $N/2$ 2-point transforms.[9]

Cooley–Tukey algorithms for general composite N are obtained by first doing the N_1-point transforms and then again applying Eq. (9.36) to another remaining factor N_2 of N/N_1 until all the factors of N have been used. The repeated application of Eq. (9.36)

[9]For Figure 9.4 to be an exact representation of Eq. (9.36), the two-point butterflies of the last stage must be replaced by the butterflies of Figure 9.10.

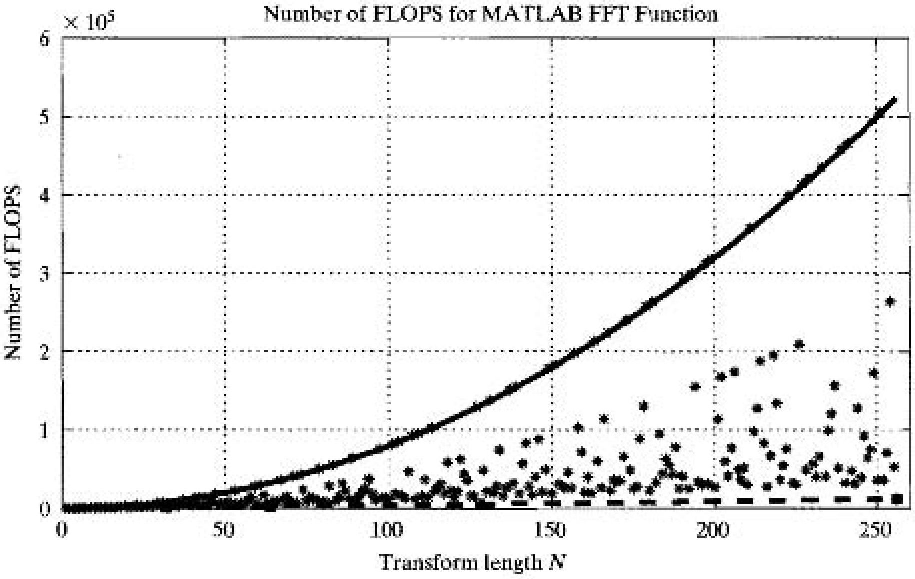


Figure 9.26 Number of floating-point operations as a function of N for MATLAB `fft( )` function (revision 5.2).

leads to decompositions similar to the power-of-two algorithms. These algorithms require only slightly more complicated indexing than the power of 2 case. If the factors of N are relatively prime, the number of multiplications can be further reduced at the cost of more complicated indexing. The "prime factor" algorithms use different index decompositions from those of Eqs. (9.35a) and (9.35b) so as to eliminate the twiddle factors in Eq. (9.36), and thus save a significant amount of computation. The details of the more general Cooley–Tukey and prime factor algorithms are discussed in Burrus and Parks (1985), Burrus (1988), and Blahut (1985).

As an illustration of what can be achieved using such prime factor algorithms, consider the measurements plotted in Figure 9.26. These measurements of the number of floating-point operations (FLOPS) as a function of N are for MATLAB's `fft( )` function in Rev. 5.2 of MATLAB.[10] As we have discussed, the total number of floating point operations should be proportional to $N \log_2 N$ for N a power of two and proportional to N^2 for direct computation. For other values of N the total operation count will be dependent on the number (and cardinality) of the factors.

When N is a prime number, direct evaluation is required so the number of FLOPS will be proportional to N^2. The upper (solid) curve in Figure 9.26 shows the function

$$\text{FLOPS}(N) = 6N^2 + 2N(N-1). \tag{9.37}$$

All the points falling on this curve are for values N a prime number. The lower dashed curve shows the function

$$\text{FLOPS}(N) = 6N \log_2 N. \tag{9.38}$$

[10]This graph was created with a modified version of a program written by C. S. Burrus. Since it is no longer possible to measure the number of floating-point operations in recent revisions of MATLAB, the reader may not be able to repeat this experiment.

The points falling on this curve are all for N a power of two. For other composite numbers the number of operations falls between the two curves. To see how efficiency varies from integer to integer, consider values of N from 199 to 202. The number 199 is a prime, so the number of operations (318004) falls on the maximum curve. The value $N = 200$ has the factorization $N = 2 \cdot 2 \cdot 2 \cdot 5 \cdot 5$, and the number of operations (27134) is near the minimum curve. For $N = 201 = 3 \cdot 67$, the number of FLOPS is 113788, and for $N = 202 = 2 \cdot 101$ the number is 167676. This wide difference between $N = 201$ and $N = 202$ is because a 101-point transform requires much more computation than a 67-point transform. Also note that when N has many small factors (such as $N = 200$) the efficiency is much greater.

9.5.2 Optimized FFT Algorithms

An FFT algorithm is based on the mathematical decomposition of the DFT into a combination of smaller transforms as we showed in detail in Sections 9.2 and 9.3. The FFT algorithm can be expressed in a high-level programming language that can be translated into machine-level instructions by compilers running on the target machine. In general, this will lead to implementations whose efficiency will vary with machine architecture. To address the issue of maximizing efficiency over a range of machines, Frigo and Johnson (1998 and 2005), developed a free-software library called FFTW ("Fastest Fourier Transform in the West"). FFTW uses a "planner" to adapt its generalized Coley–Tukey-type FFT algorithms to a given hardware platform, thereby maximizing efficiency. The system operates in two stages, the first being a planning stage in which the computations are organized so as to optimize performance on the given machine, and the second being a computation stage where the resulting plan (program) is executed. Once the plan is determined for a given machine, it can be executed on that machine as many times as needed. The details of FFTW are beyond our scope here. However, Frigo and Johnson, 2005 have shown that over a wide range of host machines, the FFTW algorithm is significantly faster than other implementations for values of N ranging from about 16 up to 8192. Above 8192, the performance of FFTW drops drastically due to memory cache issues.

9.6 IMPLEMENTATION OF THE DFT USING CONVOLUTION

Because of the dramatic efficiency of the FFT, convolution is often implemented by explicitly computing the inverse DFT of the product of the DFTs of each sequence to be convolved, where an FFT algorithm is used to compute both the forward and the inverse DFTs. In contrast, and even in apparent (but, of course, not actual) contradiction, it is sometimes preferable to compute the DFT by first reformulating it as a convolution. We have already seen an example of this in the Goertzel algorithm. A number of other, more sophisticated, procedures are based on this approach as discussed in the following sections.

9.6.1 Overview of the Winograd Fourier Transform Algorithm

One procedure proposed and developed by S. Winograd (1978), often referred to as the Winograd Fourier transform algorithm (WFTA), achieves its efficiency by expressing the DFT in terms of polynomial multiplication or, equivalently, convolution. The WFTA uses an indexing scheme corresponding to the decomposition of the DFT into a multiplicity of short-length DFTs where the lengths are relatively prime. Then the short DFTs are converted into periodic convolutions. A scheme for converting a DFT into a convolution when the number of input samples is prime was proposed by Rader (1968), but its application awaited the development of efficient methods for computing periodic convolutions. Winograd combined all of the foregoing procedures together with highly efficient algorithms for computing cyclic convolutions into a new approach to computing the DFT. The techniques for deriving efficient algorithms for computing short convolutions are based on relatively advanced number-theoretic concepts, such as the Chinese remainder theorem for polynomials, and consequently, we do not explore the details here. However, excellent discussions of the details of the WFTA are available in McClellan and Rader (1979), Blahut (1985), and Burrus (1988).

With the WFTA approach, the number of multiplications required for an N-point DFT is proportional to N rather than $N \log N$. Although this approach leads to algorithms that are optimal in terms of minimizing multiplications, the number of additions is significantly increased in comparison with the FFT. Therefore, the WFTA is most advantageous when multiplication is significantly slower than addition, as is often the case with fixed-point digital arithmetic. However, in processors where multiplication and accumulation are tied together, the Cooley–Tukey or prime factor algorithms are generally preferable. Additional difficulties with the WFTA are that indexing is more complicated, in-place computation is not possible, and there are major structural differences in algorithms for different values of N.

Thus, although the WFTA is extremely important as a benchmark for determining how efficient the DFT computation can be (in terms of number of multiplications), other factors often dominate in determining the speed and efficiency of a hardware or software implementation of the DFT computation.

9.6.2 The Chirp Transform Algorithm

Another algorithm based on expressing the DFT as a convolution is referred to as the chirp transform algorithm (CTA). This algorithm is not optimal in minimizing any measure of computational complexity, but it has been useful in a variety of applications, particularly when implemented in technologies that are well suited to doing convolution with a fixed, prespecified impulse response. The CTA is also more flexible than the FFT, since it can be used to compute *any* set of equally spaced samples of the Fourier transform on the unit circle.

To derive the CTA, we let $x[n]$ denote an N-point sequence and $X(e^{j\omega})$ its Fourier transform. We consider the evaluation of M samples of $X(e^{j\omega})$ that are equally spaced in angle on the unit circle, as indicated in Figure 9.27, i.e., at frequencies

$$\omega_k = \omega_0 + k\Delta\omega, \qquad k = 0, 1, \ldots, M-1, \tag{9.39}$$

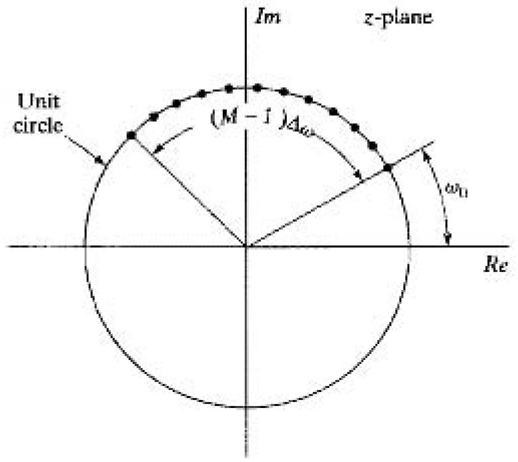


Figure 9.27 Frequency samples for chirp transform algorithm.

where the starting frequency ω_0 and the frequency increment $\Delta\omega$ can be chosen arbitrarily. (For the specific case of the DFT, $\omega_0 = 0$, $M = N$, and $\Delta\omega = 2\pi/N$.) The Fourier transform corresponding to this more general set of frequency samples is given by

$$X(e^{j\omega_k}) = \sum_{n=0}^{N-1} x[n]e^{-j\omega_k n}, \qquad k = 0, 1, \ldots, M-1, \tag{9.40}$$

or, with W defined as

$$W = e^{-j\Delta\omega} \tag{9.41}$$

and using Eq. (9.39),

$$X(e^{j\omega_k}) = \sum_{n=0}^{N-1} x[n]e^{-j\omega_0 n} W^{nk}. \tag{9.42}$$

To express $X(e^{j\omega_k})$ as a convolution, we use the identity

$$nk = \tfrac{1}{2}[n^2 + k^2 - (k-n)^2] \tag{9.43}$$

to express Eq. (9.42) as

$$X(e^{j\omega_k}) = \sum_{n=0}^{N-1} x[n]e^{-j\omega_0 n} W^{n^2/2} W^{k^2/2} W^{-(k-n)^2/2}. \tag{9.44}$$

Letting

$$g[n] = x[n]e^{-j\omega_0 n} W^{n^2/2}, \tag{9.45}$$

we can then write

$$X(e^{j\omega_k}) = W^{k^2/2}\left(\sum_{n=0}^{N-1} g[n]W^{-(k-n)^2/2}\right), \qquad k = 0, 1, \ldots, M-1. \tag{9.46}$$

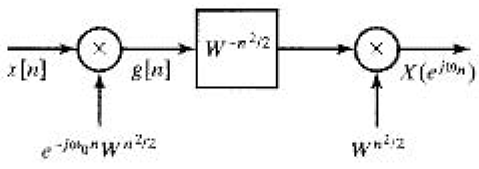


Figure 9.28 Block diagram of chirp transform algorithm.

In preparation for interpreting Eq. (9.46) as the output of a linear time-invariant system, we obtain more familiar notation by replacing k by n and n by k in Eq. (9.46):

$$X(e^{j\omega_n}) = W^{n^2/2}\left(\sum_{k=0}^{N-1} g[k]W^{-(n-k)^2/2}\right), \qquad n = 0, 1, \ldots, M-1. \tag{9.47}$$

In the form of Eq. (9.47), $X(e^{j\omega_n})$ corresponds to the convolution of the sequence $g[n]$ with the sequence $W^{-n^2/2}$, followed by multiplication by the sequence $W^{n^2/2}$. The output sequence, indexed on the independent variable n, is the sequence of frequency samples $X(e^{j\omega_n})$. With this interpretation, the computation of Eq. (9.47) is as depicted in Figure 9.28. The sequence $W^{-n^2/2}$ can be thought of as a complex exponential sequence with linearly increasing frequency $n\Delta w$. In radar systems, such signals are called chirp signals—hence the name *chirp transform*. A system similar to Figure 9.28 is commonly used in radar and sonar signal processing for pulse compression (Skolnik, 2002).

For the evaluation of the Fourier transform samples specified in Eq. (9.47), we need only compute the output of the system in Figure 9.28 over a finite interval. In Figure 9.29, we depict illustrations of the sequences $g[n]$, $W^{-n^2/2}$, and $g[n] * W^{-n^2/2}$. Since $g[n]$ is of finite duration, only a finite portion of the sequence $W^{-n^2/2}$ is used in obtaining $g[n] * W^{-n^2/2}$ over the interval $n = 0, 1, \ldots, M-1$, specifically, that portion from $n = -(N-1)$ to $n = M-1$. Let us define

$$h[n] = \begin{cases} W^{-n^2/2}, & -(N-1) \le n \le M-1, \\ 0, & \text{otherwise}, \end{cases} \tag{9.48}$$

as illustrated in Figure 9.30. It is easily verified by considering the graphical representation of the process of convolution that

$$g[n] * W^{-n^2/2} = g[n] * h[n], \qquad n = 0, 1, \ldots, M-1. \tag{9.49}$$

Consequently, the infinite-duration impulse response $W^{-n^2/2}$ in the system of Figure 9.28 can be replaced by the finite-duration impulse response of Figure 9.30. The system is now as indicated in Figure 9.31, where $h[n]$ is specified by Eq. (9.48) and the frequency samples are given by

$$X(e^{j\omega_n}) = y[n], \qquad n = 0, 1, \ldots, M-1. \tag{9.50}$$

Evaluation of frequency samples using the procedure indicated in Figure 9.31 has a number of potential advantages. In general, we do not require $N = M$ as in the FFT algorithms, and neither N nor M need be composite numbers. In fact, they may be prime numbers if desired. Furthermore, the parameter ω_0 is arbitrary. This increased flexibility over the FFT does not preclude efficient computation, since the convolution in Figure 9.31 can be implemented efficiently using an FFT algorithm with the technique

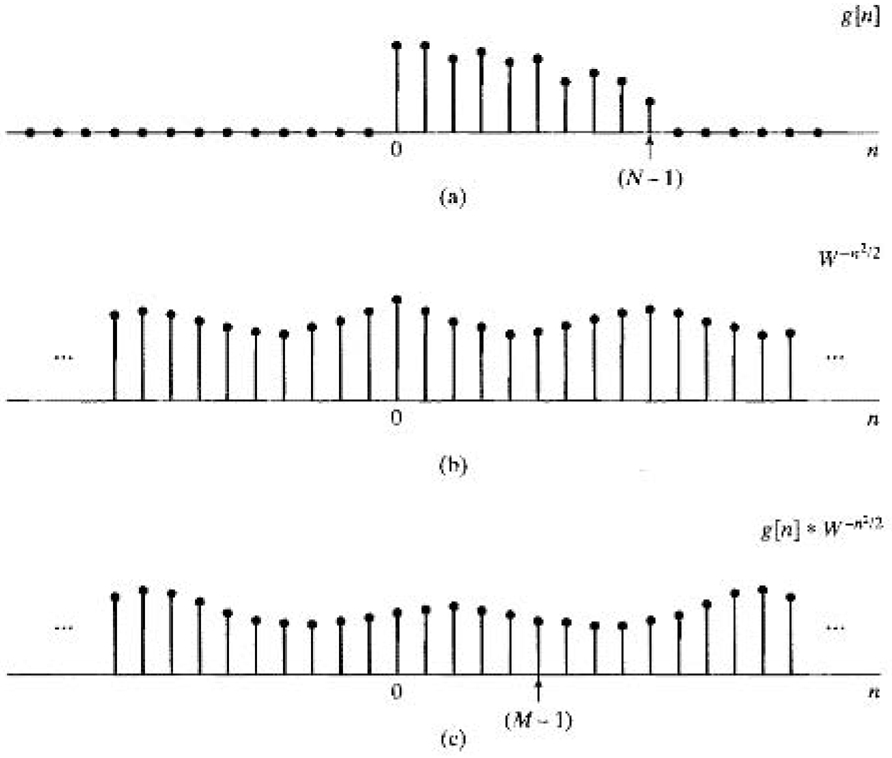


Figure 9.29 An illustration of the sequences used in the chirp transform algorithm. Note that the actual sequences involved are complex valued. (a) $g[n] = x[n]e^{-j\omega_0 n}W^{n^2/2}$. (b) $W^{-n^2/2}$. (c) $g[n] * W^{-n^2/2}$.

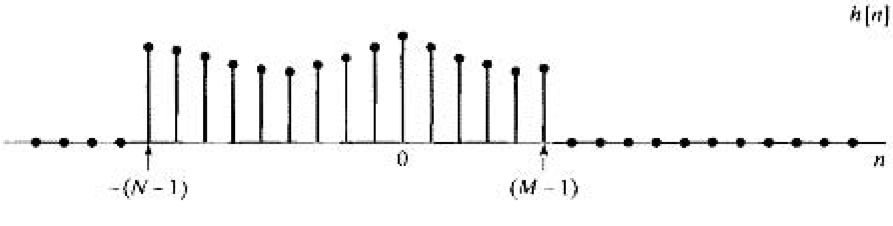


Figure 9.30 An illustration of the region of support for the FIR chirp filter. Note that the actual values of $h[n]$ as given by Eq. (9.48) are complex.

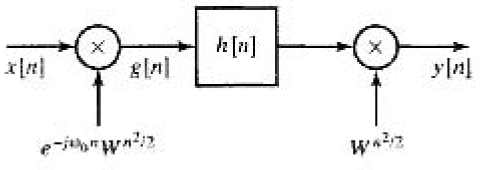


Figure 9.31 Block diagram of chirp transform system for finite-length impulse response.

of Section 8.7 to compute the convolution. As discussed in that section, the FFT size must be greater than or equal to $(M + N - 1)$ in order that the circular convolution will be equal to $g[n] * h[n]$ for $0 \le n \le M - 1$. The FFT size is otherwise arbitrary and can, for example, be chosen to be a power of 2. It is interesting to note that the FFT algorithms used to compute the convolution implied by the CTA could be of the Winograd type. These algorithms themselves use convolution to implement the DFT computation.

In the system of Figure 9.31 $h[n]$ is noncausal, and for certain real-time implementations it must be modified to obtain a causal system. Since $h[n]$ is of finite duration, this modification is easily accomplished by delaying $h[n]$ by $(N - 1)$ to obtain a causal impulse response:

$$h_1[n] = \begin{cases} W^{-(n-N+1)^2/2}, & n = 0, 1, \ldots, M + N - 2, \\ 0, & \text{otherwise}. \end{cases} \tag{9.51}$$

Since both the chirp demodulation factor at the output and the output signal are also delayed by $(N - 1)$ samples, the Fourier transform values are

$$X(e^{j\omega_n}) = y_1[n + N - 1], \qquad n = 0, 1, \ldots, M - 1. \tag{9.52}$$

Modifying the system of Figure 9.31 to obtain a causal system results in the system of Figure 9.32. An advantage of this system stems from the fact that it involves the convolution of the input signal (modulated with a chirp) with a fixed, causal impulse response. Certain technologies, such as charge-coupled devices (CCD) and surface acoustic wave (SAW) devices, are particularly useful for implementing convolution with a fixed, prespecified impulse response. These devices can be used to implement FIR filters, with the filter impulse response being specified at the time of fabrication by a geometric pattern of electrodes. A similar approach was followed by Hewes, Broderson and Buss (1979) in implementing the CTA with CCDs.

Further simplification of the CTA results when the frequency samples to be computed correspond to the DFT, i.e., when $\omega_0 = 0$ and $W = e^{-j2\pi/N}$, so that $\omega_n = 2\pi n/N$. In this case, it is convenient to modify the system of Figure 9.32. Specifically, with $\omega_0 = 0$ and $W = e^{-j2\pi/N} = W_N$, consider applying an additional unit of delay to the impulse response in Figure 9.32. With N even, $W_N^N = e^{j2\pi} = 1$, so

$$W_N^{-(n-N)^2/2} = W_N^{-n^2/2}. \tag{9.53}$$

Therefore, the system now is as shown in Figure 9.33, where

$$h_2[n] = \begin{cases} W_N^{-n^2/2}, & n = 1, 2, \ldots, M + N - 1, \\ 0, & \text{otherwise}. \end{cases} \tag{9.54}$$

In this case, the chirp signal modulating $x[n]$ and the chirp signal modulating the output of the FIR filter are identical, and

$$X(e^{j2\pi n/N}) = y_2[n + N], \qquad n = 0, 1, \ldots, M - 1. \tag{9.55}$$

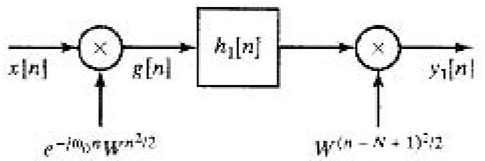


Figure 9.32 Block diagram of chirp transform system for causal finite-length impulse response.

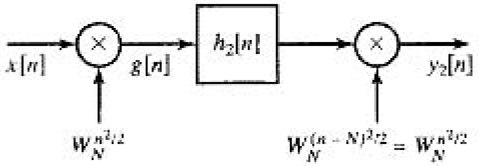


Figure 9.33 Block diagram of chirp transform system for obtaining DFT samples.

Example 9.1 Chirp Transform Parameters

Suppose we have a finite-length sequence $x[n]$ that is nonzero only on the interval $n = 0, \ldots, 25$, and we wish to compute 16 samples of the DTFT $X(e^{j\omega})$ at the frequencies $\omega_k = 2\pi/27 + 2\pi k/1024$ for $k = 0, \ldots, 15$. We can compute the desired frequency samples through convolution with a causal impulse response using the system in Figure 9.32 with an appropriate choice of parameters. We set $M = 16$, the number of samples desired, and $N = 26$, the length of the sequence. The frequency of the initial sample, ω_0, is $2\pi/27$, while the interval between adjacent frequency samples, $\Delta\omega$, is $2\pi/1024$. With these choices for the parameters, we know from Eq. (9.41) that $W = e^{-j\Delta\omega}$, and so the causal impulse response we desire is from Eq. (9.51)

$$h_1[n] = \begin{cases} [e^{-j2\pi/1024}]^{-(n-25)^2/2}, & n = 0, \ldots, 40, \\ 0, & \text{otherwise.} \end{cases}$$

For this causal impulse response, the output $y_1[n]$ will be the desired frequency samples beginning at $y_1[25]$, i.e.,

$$y_1[n+25] = X(e^{j\omega_n})|_{\omega_n = 2\pi/27 + 2\pi n/1024}, \qquad n = 0, \ldots, 15.$$

An algorithm similar to the CTA was first proposed by Bluestein (1970), who showed that a recursive realization of Figure 9.32 can be obtained for the case $\Delta\omega = 2\pi/N$, N a perfect square. (See Problem 9.48.) Rabiner, Schafer and Rader (1969) generalized this algorithm to obtain samples of the z-transform equally spaced in angle on a spiral contour in the z-plane. This more general form of the CTA was called the chirp z-transform (CZT) algorithm. The algorithm that we have called the CTA is a special case of the CZT algorithm.

9.7 EFFECTS OF FINITE REGISTER LENGTH

Since the fast Fourier transform algorithm is widely used for digital filtering and spectrum analysis, it is important to understand the effects of finite register length in the computation. As in the case of digital filters, a precise analysis of the effects is difficult. However, a simplified analysis is often sufficient for the purpose of choosing the required register length. The analysis that we will present is similar in style to that carried out in Section 6.9. Specifically, we analyze arithmetic round-off by means of a linear-noise model obtained by inserting an additive noise source at each point in the computation algorithm where round-off occurs. Furthermore, we will make a number of assumptions to simplify the analysis. The results that we obtain lead to several simplified, but useful, estimates of the effect of arithmetic round-off. Although the analysis is for rounding, it is generally easy to modify the results for truncation.

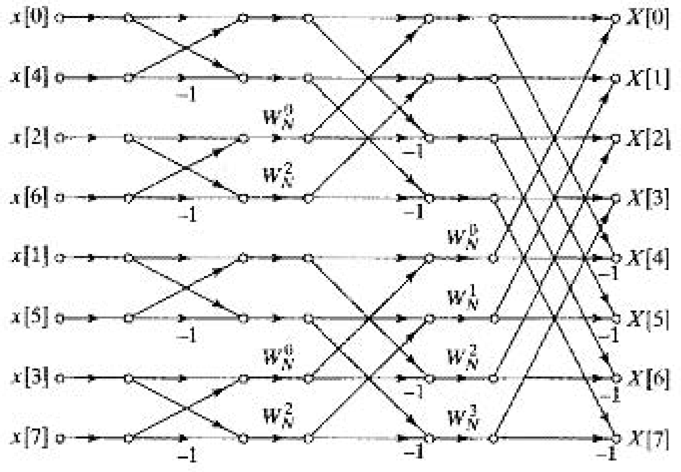


Figure 9.34 Flow graph for decimation-in-time FFT algorithm.

We have seen several different algorithmic structures for the FFT. However, the effects of round-off noise are very similar among the different classes of algorithms. Therefore, even though we consider only the radix-2 decimation-in-time algorithm, our results are representative of other forms as well.

The flow graph depicting a decimation-in-time algorithm for $N = 8$ was shown in Figure 9.11 and is reproduced in Figure 9.34. Some key aspects of this diagram are common to all standard radix-2 algorithms. The DFT is computed in $\nu = \log_2 N$ stages. At each stage a new array of N numbers is formed from the previous array by linear combinations of the elements, taken two at a time. The ν^{th} array contains the desired DFT. For radix-2 decimation-in-time algorithms, the basic 2-point DFT computation is of the form

$$X_m[p] = X_{m-1}[p] + W_N^r X_{m-1}[q], \tag{9.56a}$$

$$X_m[q] = X_{m-1}[p] - W_N^r X_{m-1}[q]. \tag{9.56b}$$

Here the subscripts m and $(m-1)$ refer to the m^{th} array and the $(m-1)^{\text{st}}$ array, respectively, and p and q denote the location of the numbers in each array. (Note that $m = 0$ refers to the input array and $m = \nu$ refers to the output array.) A flow graph representing the butterfly computation is shown in Figure 9.35.

At each stage, $N/2$ separate butterfly computations are carried out to produce the next array. The integer r varies with p, q, and m in a manner that depends on the specific form of the FFT algorithm used. However, our analysis is not tied to the specific way

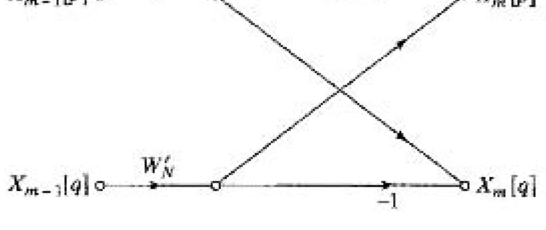


Figure 9.35 Butterfly computation for decimation-in-time.

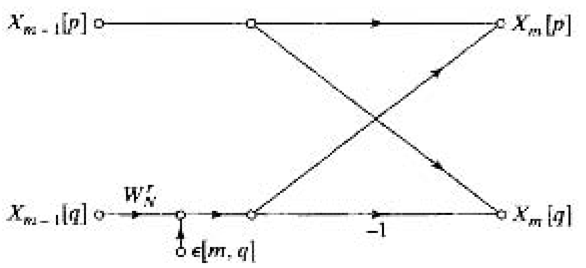


Figure 9.36 Linear-noise model for fixed-point round-off noise in a decimation-in-time butterfly computation.

in which r varies. Also, the specific relationship among p, q, and m, which determines how we index through the m^{th} array, is not important for the analysis. The details of the analysis for decimation in time and decimation in frequency differ somewhat due to the different butterfly forms, but the basic results do not change significantly. In our analysis we assume a butterfly of the form of Eqs. (9.56a) and (9.56b), corresponding to decimation in time.

We model the round-off noise by associating an additive noise generator with each fixed-point multiplication. With this model, the butterfly of Figure 9.35 is replaced by that of Figure 9.36 for analyzing the round-off noise effects. The notation $\varepsilon[m,q]$ represents the complex-valued error introduced in computing the m^{th} array from the $(m-1)^{\text{st}}$ array; specifically, it indicates the error resulting from quantization of multiplication of the q^{th} element of the $(m-1)^{\text{st}}$ array by a complex coefficient.

Since we assume that, in general, the input to the FFT is a complex sequence, each of the multiplications is complex and thus consists of four real multiplications. We assume that the errors due to each real multiplication have the following properties:

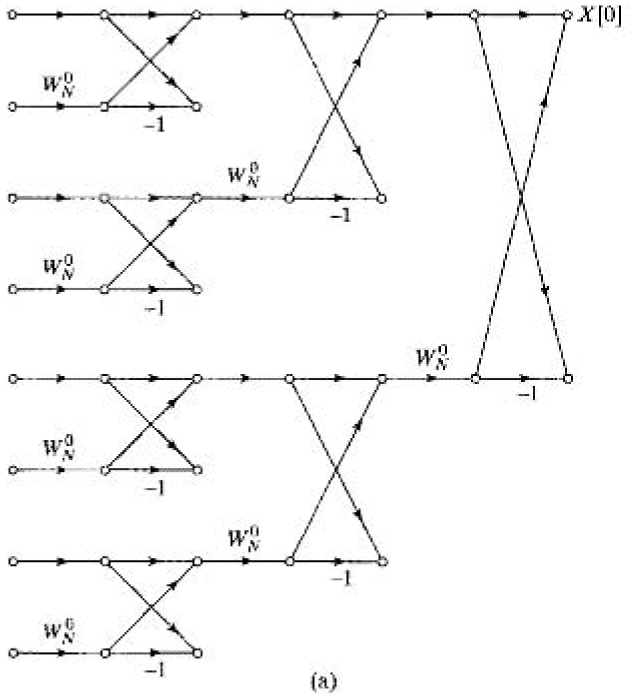
1. The errors are uniformly distributed random variables over the range $-(1/2)\cdot 2^{-B}$ to $(1/2)\cdot 2^{-B}$, where, as defined in Section 6.7.1, numbers are represented as $(B+1)$-bit signed fractions. Therefore, each error source has variance $2^{-2B}/12$.
2. The errors are uncorrelated with one another.
3. All the errors are uncorrelated with the input and, consequently, also with the output.

Since each of the four noise sequences is uncorrelated zero-mean white noise and all have the same variance,

$$\mathcal{E}\{|\varepsilon[m,q]|^2\} = 4\cdot\frac{2^{-2B}}{12} = \tfrac{1}{3}\cdot 2^{-2B} = \sigma_B^2. \tag{9.57}$$

To determine the mean-square value of the output noise at any output node, we must account for the contribution from each of the noise sources that propagate to that node. We can make the following observations from the flow graph of Figure 9.34:

1. The transmission function from any node in the flow graph to any other node to which it is connected is multiplication by a complex constant of unity magnitude (because each branch transmittance is either unity or an integer power of W_N).
2. Each output node connects to seven butterflies in the flow graph. (In general, each output node would connect to $(N-1)$ butterflies.) For example, Figure 9.37(a)



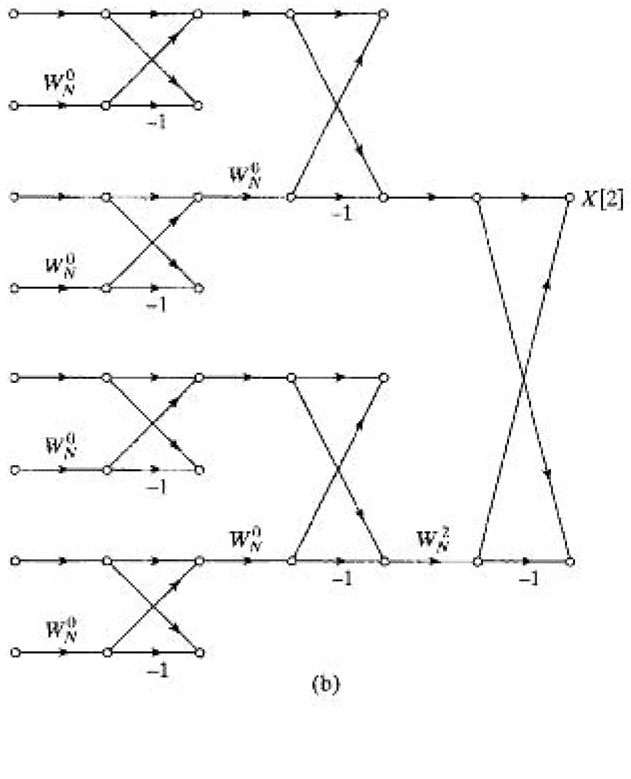


Figure 9.37 (a) Butterflies that affect $X[0]$; (b) butterflies that affect $X[2]$.

shows the flow graph with all the butterflies removed that do not connect to $X[0]$, and Figure 9.37(b) shows the flow graph with all the butterflies removed that do not connect to $X[2]$.

These observations can be generalized to the case of N an arbitrary power of 2.

As a consequence of the first observation, the mean-square value of the magnitude of the component of the output noise due to each elemental noise source is the same and equal to σ_B^2. The total output noise at each output node is equal to the sum of the noise propagated to that node. Since we assume that all the noise sources are uncorrelated, the mean-square value of the magnitude of the output noise is equal to σ_B^2 times the number of noise sources that propagate to that node. At most one complex noise source is introduced at each butterfly; consequently, from observation 2, at most $(N-1)$ noise sources propagate to each output node. In fact, not all the butterflies generate round-off noise, since some (for example, all those in the first and second stages for $N=8$) involve only multiplication by unity. However, if for simplicity we assume that round-off occurs for each butterfly, we can consider the result as an upper bound on the output noise. With this assumption, then, the mean square value of the output noise in the k^{th} DFT value, $F[k]$, is given by

$$\mathcal{E}\{|F[k]|^2\} = (N-1)\sigma_B^2, \tag{9.58}$$

which, for large N, we approximate as

$$\mathcal{E}\{|F[k]|^2\} \cong N\sigma_B^2. \tag{9.59}$$

According to this result, the mean-square value of the output noise is proportional to N, the number of points transformed. The effect of doubling N, or adding another stage in the FFT, is to double the mean-square value of the output noise. In Problem 9.52, we consider the modification of this result when we do not insert noise sources for those butterflies that involve only multiplication by unity or j. Note that for FFT algorithms, a double-length accumulator does not help us reduce round-off noise, since the outputs of the butterfly computation must be stored in $(B+1)$-bit registers at the output of each stage.

In implementing an FFT algorithm with fixed-point arithmetic, we must ensure against overflow. From Eqs. (9.56a) and (9.56b), it follows that

$$\max(|X_{m-1}[p]|, |X_{m-1}[q]|) \le \max(|X_m[p]|, |X_m[q]|) \tag{9.60}$$

and also

$$\max(|X_m[p]|, |X_m[q]|) \le 2\max(|X_{m-1}[p]|, |X_{m-1}[q]|). \tag{9.61}$$

(See Problem 9.51.) Equation (9.60) implies that the maximum magnitude is nondecreasing from stage to stage. If the magnitude of the output of the FFT is less than unity, then the magnitude of the points in each array must be less than unity, i.e., there will be no overflow in any of the arrays.[11]

To express this constraint as a bound on the input sequence, we note that the condition

$$|x[n]| < \frac{1}{N}, \qquad 0 \le n \le N-1, \tag{9.62}$$

[11] Actually, one should discuss overflow in terms of the real and imaginary parts of the data rather than the magnitude. However, $|x| < 1$ implies that $|\mathcal{R}e\{x\}| < 1$ and $|\mathcal{I}m\{x\}| < 1$, and only a slight increase in allowable signal level is achieved by scaling on the basis of real and imaginary parts.

is both necessary and sufficient to guarantee that

$$|X[k]| < 1, \qquad 0 \le k \le N-1. \tag{9.63}$$

This follows from the definition of the DFT, since

$$|X[k]| = \left|\sum_{n=0}^{N-1} x[n]W_N^{kn}\right| \le \sum_{n=0}^{N-1} |x[n]| \quad k = 0, 1, \ldots N-1. \tag{9.64}$$

Thus, Eq. (9.62) is sufficient to guarantee that there will be no overflow for all stages of the algorithm.

To obtain an explicit expression for the noise-to-signal ratio at the output of the FFT algorithm, consider an input in which successive sequence values are uncorrelated, i.e., a white-noise input signal. Also, assume that the real and imaginary parts of the input sequence are uncorrelated and that each has an amplitude density that is uniform between $-1/(\sqrt{2}N)$ and $+1/(\sqrt{2}N)$. (Note that this signal satisfies Eq. (9.62).) Then the average squared magnitude of the complex input sequence is

$$\mathcal{E}\{|x[n]|^2\} = \frac{1}{3N^2} = \sigma_x^2. \tag{9.65}$$

The DFT of the input sequence is

$$X[k] = \sum_{n=0}^{N-1} x[n]W^{kn}, \tag{9.66}$$

from which it can be shown that, under the foregoing assumptions on the input,

$$\begin{aligned} \mathcal{E}\{|X[k]|^2\} &= \sum_{n=0}^{N-1} \mathcal{E}\{|x[n]|^2\}|W^{kn}|^2 \\ &= N\sigma_x^2 = \frac{1}{3N}. \end{aligned} \tag{9.67}$$

Combining Eqs. (9.59) and (9.67), we obtain

$$\frac{\mathcal{E}\{|F[k]|^2\}}{\mathcal{E}\{|X[k]|^2\}} = 3N^2\sigma_B^2 = N^2 2^{-2B}. \tag{9.68}$$

According to Eq. (9.68), the noise-to-signal ratio increases as N^2, or 1 bit per stage. That is, if N is doubled, corresponding to adding one additional stage to the FFT, then to maintain the same noise-to-signal ratio, 1 bit must be added to the register length. The assumption of a white-noise input signal is, in fact, not critical here. For a variety of other inputs, the noise-to-signal ratio is still proportional to N^2, with only the constant of proportionality changing.

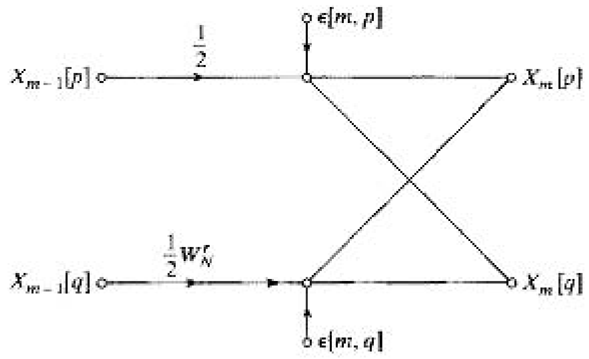


Figure 9.38 Butterfly showing scaling multipliers and associated fixed-point round-off noise.

Equation (9.61) suggests an alternative scaling procedure. Since the maximum magnitude increases by no more than a factor of 2 from stage to stage, we can prevent overflow by requiring that $|x[n]| < 1$ and incorporating an attenuation of $\frac{1}{2}$ at the input to each stage. In this case, the output will consist of the DFT scaled by $1/N$. Although the mean-square output signal will be $1/N$ times what it would be if no scaling were introduced, the input amplitude can be N times larger without causing overflow. For the white-noise input signal, this means that we can assume that the real and imaginary parts are uniformly distributed from $-1/\sqrt{2}$ to $1/\sqrt{2}$, so that $|x[n]| < 1$. Thus, with the ν divisions by 2, the maximum expected value of the magnitude squared of the DFT that can be attained (for the white input signal) is the same as that given in Eq. (9.67). However, the output noise level will be much less than in Eq. (9.59), since the noise introduced at early stages of the FFT will be attenuated by the scaling that takes place in the later arrays. Specifically, with scaling by 1/2 introduced at the input to each butterfly, we modify the butterfly of Figure 9.36 to that of Figure 9.38, where, in particular, two noise sources are now associated with each butterfly. As before, we assume that the real and imaginary parts of these noise sources are uncorrelated and are also uncorrelated with the other noise sources and that the real and imaginary parts are uniformly distributed between $\pm(1/2)\cdot 2^{-B}$. Thus, as before,

$$\mathcal{E}\{|\varepsilon[m, q]|^2\} = \sigma_B^2 = \tfrac{1}{3}\cdot 2^{-2B} = \mathcal{E}\{|\varepsilon[m, p]|^2\}. \qquad (9.69)$$

Because the noise sources are all uncorrelated, the mean-squared magnitude of the noise at each output node is again the sum of the mean-squared contributions of each noise source in the flow graph. However, unlike the previous case, the attenuation that each noise source experiences through the flow graph depends on the array at which it originates. A noise source originating at the m^{th} array will propagate to the output with multiplication by a complex constant with magnitude $(1/2)^{\nu-m-1}$. By examination of Figure 9.34, we see that for the case $N = 8$, each output node connects to:

1 butterfly originating at the $(\nu-1)^{\text{st}}$ array,

2 butterflies originating at the $(\nu-2)^{\text{nd}}$ array,

4 butterflies originating at the $(\nu-3)^{\text{rd}}$ array, etc.

For the general case with $N = 2^\nu$, each output node connects to $2^{\nu-m-1}$ butterflies and therefore to $2^{\nu-m}$ noise sources that originate at the m^{th} array. Thus, at each output node, the mean-square magnitude of the noise is

$$\begin{aligned}\mathcal{E}\{|F[k]|^2\} &= \sigma_B^2 \sum_{m=0}^{\nu-1} 2^{\nu-m} \cdot (0.5)^{2\nu-2m-2} \\ &= \sigma_B^2 \sum_{m=0}^{\nu-1} (0.5)^{\nu-m-2} \\ &= \sigma_B^2 \cdot 2 \sum_{k=0}^{\nu-1} 0.5^k \\ &= 2\sigma_B^2 \frac{1-0.5^\nu}{1-0.5} = 4\sigma_B^2(1-0.5^\nu).\end{aligned} \tag{9.70}$$

For large N, we assume that 0.5^ν (i.e., $1/N$) is negligible compared with unity, so

$$\mathcal{E}\{|F[k]|^2\} \cong 4\sigma_B^2 = \tfrac{4}{3} \cdot 2^{-2B}, \tag{9.71}$$

which is much less than the noise variance resulting when all the scaling is carried out on the input data.

Now we can combine Eq. (9.71) with Eq. (9.67) to obtain the output noise-to-signal ratio for the case of step-by-step scaling and white input. We obtain

$$\frac{\mathcal{E}\{|F[k]|^2\}}{\mathcal{E}\{|X[k]|^2\}} = 12N\sigma_B^2 = 4N \cdot 2^{-2B}, \tag{9.72}$$

a result proportional to N rather than to N^2. An interpretation of Eq. (9.72) is that the output noise-to-signal ratio increases as N, corresponding to half a bit per stage, a result first obtained by Welch (1969). It is important to note again that the assumption of a white-noise signal is not essential in the analysis. The basic result of an increase of half a bit per stage holds for a broad class of signals, with only the constant multiplier in Eq. (9.72) being dependent on the signal.

We should also note that the dominant factor that causes the increase of the noise-to-signal ratio with N is the decrease in signal level (required by the overflow constraint) as we pass from stage to stage. According to Eq. (9.71), very little noise (only a bit or two) is present in the final array. Most of the noise has been shifted out of the binary word by the scalings.

We have assumed straight fixed-point computation in the preceding discussion; i.e., only preset attenuations were allowed, and we were not permitted to rescale on the basis of an overflow test. Clearly, if the hardware or programming facility is such that straight fixed-point computation must be used, we should, if possible, incorporate attenuators of $1/2$ at each array rather than use a large attenuation of the input array.

A third approach to avoiding overflow is the use of *block floating point*. In this procedure the original array is normalized to the far left of the computer word, with the restriction that $|x[n]| < 1$; the computation proceeds in a fixed-point manner, except that after every addition there is an overflow test. If overflow is detected, the entire array is divided by 2 and the computation continues. The number of necessary divisions by 2 are counted to determine a scale factor for the entire final array. The output noise-to-signal ratio depends strongly on how many overflows occur and at what stages of the computation they occur. The positions and timing of overflows are determined by the signal being transformed; thus, to analyze the noise-to-signal ratio in a block floating-point implementation of the FFT, we would need to know the input signal.

The preceding analysis shows that scaling to avoid overflow is the dominant factor in determining the noise-to-signal ratio of fixed-point implementations of FFT algorithms. Therefore, floating-point arithmetic should improve the performance of these algorithms. The effect of floating-point round-off on the FFT was analyzed both theoretically and experimentally by Gentleman and Sande (1966), Weinstein and Oppenheim (1969), and Kaneko and Liu (1970). These investigations show that, since scaling is no longer necessary, the decrease of noise-to-signal ratio with increasing N is much less dramatic than for fixed-point arithmetic.

For example, Weinstein (1969) showed theoretically that the noise-to-signal ratio is proportional to ν for $N = 2^\nu$, rather than proportional to N as in the fixed-point case. Therefore, quadrupling ν (raising N to the fourth power) increases the noise-to-signal ratio by only 1 bit.

9.8 SUMMARY

In this chapter we have considered techniques for computation of the discrete Fourier transform, and we have seen how the periodicity and symmetry of the complex factor $e^{-j(2\pi/N)kn}$ can be exploited to increase the efficiency of DFT computations.

We considered the Goertzel algorithm and the direct evaluation of the DFT expression because of the importance of these techniques when not all N of the DFT values are required. However, our major emphasis was on fast Fourier transform (FFT) algorithms. We described the decimation-in-time and decimation-in-frequency classes of FFT algorithms in some detail and some of the implementation considerations, such as indexing and coefficient quantization. Much of the detailed discussion concerned algorithms that require N to be a power of 2, since these algorithms are easy to understand, simple to program, and most often used.

The use of convolution as the basis for computing the DFT was briefly discussed. We presented a brief overview of the Winograd Fourier transform algorithm, and in somewhat more detail we discussed an algorithm called the chirp transform algorithm.

The final section of the chapter discussed effects of finite word length in DFT computations. We used linear-noise models to show that the noise-to-signal ratio of a DFT computation varies differently with the length of the sequence, depending on how scaling is done. We also commented briefly on the use of floating-point representations.

Problems

Basic Problems with Answers

9.1. Suppose that a computer program is available for computing the DFT

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j(2\pi/N)kn}, \qquad k = 0, 1, \ldots, N-1;$$

i.e., the input to the program is the sequence $x[n]$ and the output is the DFT $X[k]$. Show how the input and/or output sequences may be rearranged such that the program can also be used to compute the inverse DFT

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]e^{j(2\pi/N)kn}, \qquad n = 0, 1, \ldots, N-1;$$

i.e., the input to the program should be $X[k]$ or a sequence simply related to $X[k]$, and the output should be either $x[n]$ or a sequence simply related to $x[n]$. There are several possible approaches.

9.2. Computing the DFT generally requires complex multiplications. Consider the product $X + jY = (A + jB)(C + jD) = (AC - BD) + j(BC + AD)$. In this form, a complex multiplication requires four real multiplications and two real additions. Verify that a complex multiplication can be performed with three real multiplications and five additions using the algorithm

$$X = (A - B)D + (C - D)A,$$
$$Y = (A - B)D + (C + D)B.$$

9.3. Suppose that you time-reverse and delay a real-valued 32-point sequence $x[n]$ to obtain $x_1[n] = x[32-n]$. If $x_1[n]$ is used as the input for the system in Figure P9.4, find an expression for $y[32]$ in terms of $X(e^{j\omega})$, the DTFT of the original sequence $x[n]$.

9.4. Consider the system shown in Figure P9.4. If the input to the system, $x[n]$, is a 32-point sequence in the interval $0 \le n \le 31$, the output $y[n]$ at $n = 32$ is equal to $X(e^{j\omega})$ evaluated at a specific frequency ω_k. What is ω_k for the coefficients shown in Figure P9.4?

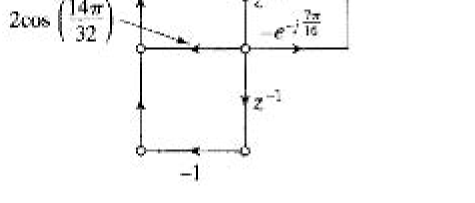


Figure P9.4

9.5. Consider the signal flow graph in Figure P9.5. Suppose that the input to the system $x[n]$ is an 8-point sequence. Choose the values of a and b such that $y[8] = X(e^{j6\pi/8})$.

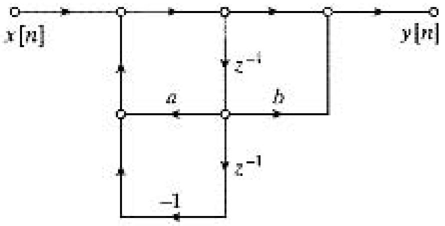


Figure P9.5

9.6. Figure P9.6 shows the graph representation of a decimation-in-time FFT algorithm for $N = 8$. The heavy line shows a path from sample $x[7]$ to DFT sample $X[2]$.

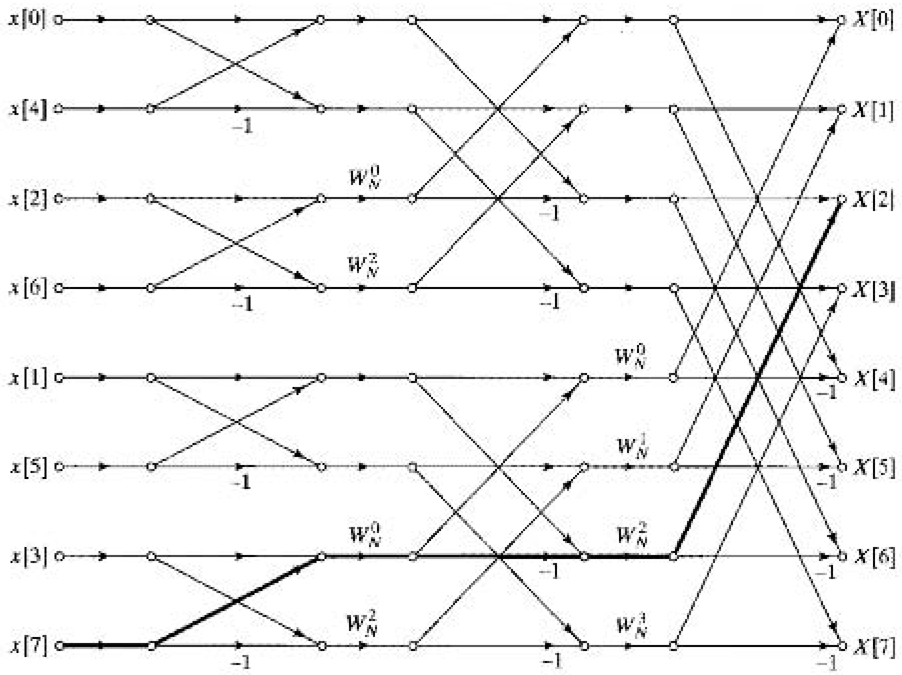


Figure P9.6

(a) What is the "gain" along the path that is emphasized in Figure P9.6?

(b) How many other paths in the flow graph begin at $x[7]$ and end at $X[2]$? Is this true in general? That is, how many paths are there between each input sample and each output sample?

(c) Now consider the DFT sample $X[2]$. By tracing paths in the flow graph of Figure P9.6, show that each input sample contributes the proper amount to the output DFT sample; i.e., verify that

$$X[2] = \sum_{n=0}^{N-1} x[n]e^{-j(2\pi/N)2n}.$$

9.7. Figure P9.7 shows the flow graph for an 8-point decimation-in-time FFT algorithm. Let $x[n]$ be the sequence whose DFT is $X[k]$. In the flow graph, $A[\cdot]$, $B[\cdot]$, $C[\cdot]$, and $D[\cdot]$ represent separate arrays that are indexed consecutively in the same order as the indicated nodes.

(a) Specify how the elements of the sequence $x[n]$ should be placed in the array $A[r]$, $r = 0, 1, \ldots, 7$. Also, specify how the elements of the DFT sequence should be extracted from the array $D[r]$, $r = 0, 1, \ldots, 7$.

(b) Without determining the values in the intermediate arrays, $B[\cdot]$ and $C[\cdot]$, determine and sketch the array sequence $D[r]$, $r = 0, 1, \ldots, 7$, if the input sequence is $x[n] = (-W_N)^n$, $n = 0, 1, \ldots, 7$.

(c) Determine and sketch the sequence $C[r]$, $r = 0, 1, \ldots, 7$, if the output Fourier transform is $X[k] = 1$, $k = 0, 1, \ldots, 7$.

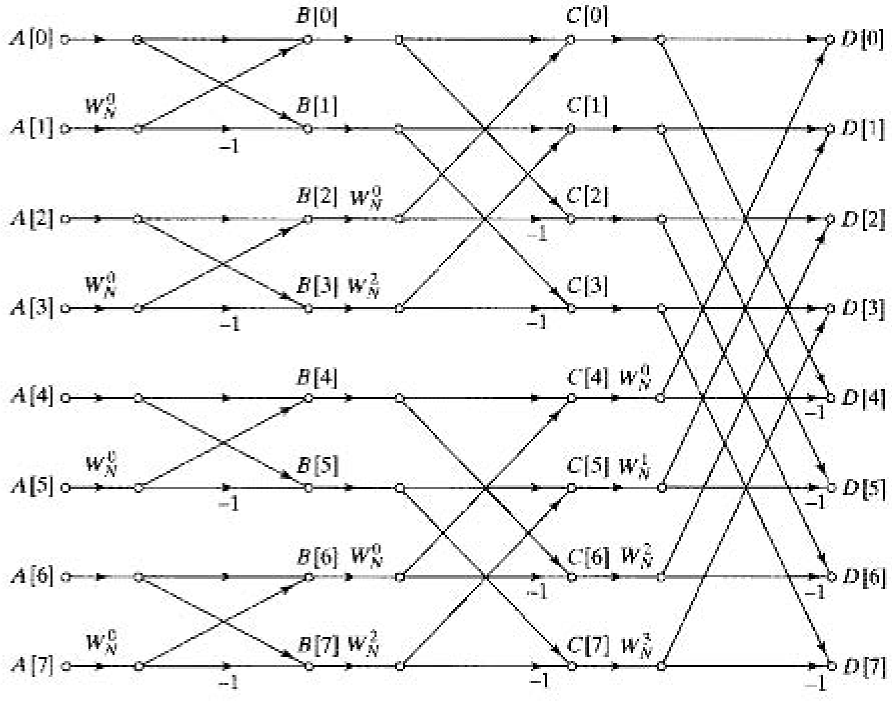


Figure P9.7

9.8. In implementing an FFT algorithm, it is sometimes useful to generate the powers of W_N with a recursive difference equation, or oscillator. In this problem we consider a radix-2 decimation-in-time algorithm for $N = 2^\nu$. Figure 9.11 depicts this type of algorithm for $N = 8$. To generate the coefficients efficiently, the frequency of the oscillator would change from stage to stage.

Assume that the arrays are numbered 0 through $\nu = \log_2 N$, so the array holding the initial input sequence is the zero$^{\text{th}}$ array and the DFT is in the ν^{th} array. In computing the butterflies in a given stage, all butterflies requiring the same coefficients W_N^r are evaluated before obtaining new coefficients. In indexing through the array, we assume that the data in the array are stored in consecutive complex registers numbered 0 through $(N-1)$. All the following questions are concerned with the computation of the m^{th} array from the $(m-1)^{\text{st}}$ array, where $1 \leq m \leq \nu$. Answers should be expressed in terms of m.

(a) How many butterflies must be computed in the m^{th} stage? How many different coefficients are required in the m^{th} stage?

(b) Write a difference equation whose impulse response $h[n]$ contains the coefficients W_N^r required by the butterflies in the m^{th} stage.

(c) The difference equation from part (b) should have the form of an oscillator, i.e., $h[n]$ should be periodic for $n \geq 0$. What is the period of $h[n]$? Based on this, write an expression for the frequency of this oscillator as a function of m.

9.9. Consider the butterfly in Figure P9.9. This butterfly was extracted from a signal flow graph implementing an FFT algorithm. Choose the most accurate statement from the following list:

1. The butterfly was extracted from a decimation-in-time FFT algorithm.
2. The butterfly was extracted from a decimation-in-frequency FFT algorithm.
3. It is not possible to say from the figure which kind of FFT algorithm the butterfly came from.

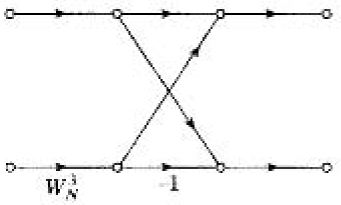


Figure P9.9

9.10. A finite-length signal $x[n]$ is nonzero in the interval $0 \leq n \leq 19$. This signal is the input to the system shown in Figure P9.10, where

$$h[n] = \begin{cases} e^{j(2\pi/21)(n-19)^2/2}, & n = 0, 1, \dots, 28, \\ 0, & \text{otherwise.} \end{cases}$$

$$W = e^{-j(2\pi/21)}$$

The output of the system, $y[n]$, for the interval $n = 19, \dots, 28$ can be expressed in terms of the DTFT $X(e^{j\omega})$ for appropriate values of ω. Write an expression for $y[n]$ in this interval in terms of $X(e^{j\omega})$.

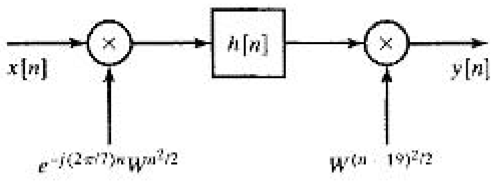


Figure P9.10

9.11. The butterfly flow graph in Figure 9.10 can be used to compute the DFT of a sequence of length $N = 2^\nu$ "in-place," i.e., using a single array of complex-valued registers. Assume this array of registers $A[\ell]$ is indexed on $0 \leq l \leq N-1$. The input sequence is initially stored in $A[\ell]$ in bit-reversed order. The array is then processed by ν stages of butterflies. Each butterfly takes two array elements $A[\ell_0]$ and $A[\ell_1]$ as inputs, then stores its outputs into those same array locations. The values of ℓ_0 and ℓ_1 depend on the stage number and the location of the butterfly in the signal flow graph. The stages of the computation are indexed by $m = 1, \ldots, \nu$.

(a) What is $|\ell_1 - \ell_0|$ as a function of the stage number m?

(b) Many stages contain butterflies with the same "twiddle" factor W_N^r. For these stages, how far apart are the values of ℓ_0 for the butterflies with the same W_N^r?

9.12. Consider the system shown in Figure P9.12, with

$$h[n] = \begin{cases} e^{j(2\pi/10)(n-11)^2/2}, & n = 0, 1, \ldots, 15, \\ 0, & \text{otherwise}. \end{cases}$$

It is desired that the output of the system, $y[n + 11] = X(e^{j\omega_n})$, where $\omega_n = (2\pi/19) + n(2\pi/10)$ for $n = 0, \ldots, 4$. Give the correct value for the sequence $r[n]$ in Figure P9.12 such that the output $y[n]$ provides the desired samples of the DTFT.

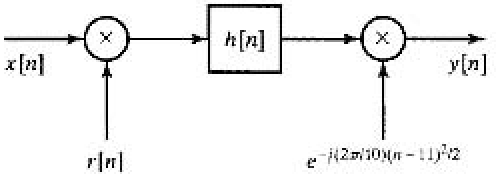


Figure P9.12

9.13. Assume that you wish to sort a sequence $x[n]$ of length $N = 16$ into bit-reversed order for input to an FFT algorithm. Give the new sample order for the bit-reversed sequence.

9.14. For the following statement, assume that the sequence $x[n]$ has length $N = 2^\nu$ and that $X[k]$ is the N-point DFT of $x[n]$. Indicate whether the statement is true or false, and justify your answer.

Statement: It is impossible to construct a signal flow graph to compute $X[k]$ from $x[n]$ such that both $x[n]$ and $X[k]$ are in normal sequential (not bit-reversed) order.

9.15. The butterfly in Figure P9.15 was taken from a decimation-in-frequency FFT with $N = 16$, where the input sequence was arranged in normal order. Note that a 16-point FFT will have four stages, indexed $m = 1, \ldots, 4$. Which of the four stages have butterflies of this form? Justify your answer.

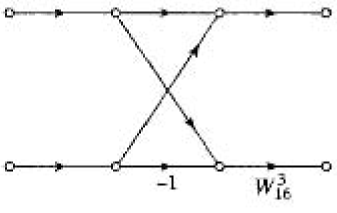


Figure P9.15

9.16. The butterfly in Figure P9.16 was taken from a decimation-in-time FFT with $N = 16$. Assume that the four stages of the signal flow graph are indexed by $m = 1, \ldots, 4$. What are the possible values of r for each of the four stages?

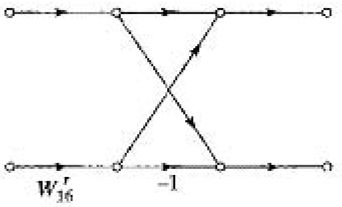


Figure P9.16

9.17. Suppose you have two programs for computing the DFT of a sequence $x[n]$ that has $N = 2^\nu$ nonzero samples. Program A computes the DFT by directly implementing the definition of the DFT sum from Eq. (8.67) and takes N^2 seconds to run. Program B implements the decimation-in-time FFT algorithm and takes $10N\log_2 N$ seconds to run. What is the shortest sequence N such that Program B runs faster than Program A?

9.18. The butterfly in Figure P9.18 was taken from a decimation-in-time FFT with $N = 16$. Assume that the four stages of the signal flow graph are indexed by $m = 1, \ldots, 4$. Which of the four stages have butterflies of this form?

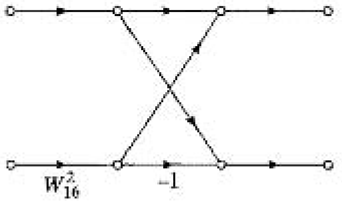


Figure P9.18

9.19. Suppose you are told that an $N = 32$ FFT algorithm has a "twiddle" factor of W_{32}^2 for one of the butterflies in its fifth (last) stage. Is the FFT a decimation-in-time or decimation-in-frequency algorithm?

9.20. Suppose you have a signal $x[n]$ with 1021 nonzero samples whose DTFT you wish to estimate by computing the DFT. You find that it takes your computer 100 seconds to compute the 1021-point DFT of $x[n]$. You then add three zero-valued samples at the end of the sequence to form a 1024-point sequence $x_1[n]$. The same program on your computer requires only 1 second to compute $X_1[k]$. Reflecting, you realize that by using $x_1[n]$, you are able to compute more samples of $X(e^{j\omega})$ in a much shorter time by adding some zeros to the end of $x[n]$ and pretending that the sequence is longer. How do you explain this apparent paradox?

Basic Problems

9.21. In Section 9.1.2, we used the fact that $W_N^{-kN} = 1$ to derive a recurrence algorithm for computing a specific DFT value $X[k]$ for a finite-length sequence $x[n]$, $n = 0, 1, \ldots, N-1$.

(a) Using the fact that $W_N^{kN} = W_N^{Nn} = 1$, show that $X[N-k]$ can be obtained as the output after N iterations of the difference equation depicted in Figure P9.21-1. That is, show that

$$X[N-k] = y_k[N].$$

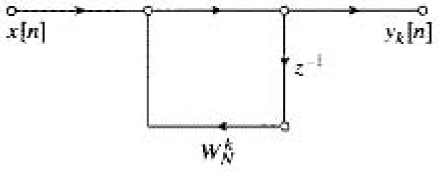


Figure P9.21-1

(b) Show that $X[N-k]$ is also equal to the output after N iterations of the difference equation depicted in Figure P9.21-2. Note that the system of Figure P9.21-2 has the same poles as the system in Figure 9.2, but the coefficient required to implement the complex zero in Figure P9.21-2 is the complex conjugate of the corresponding coefficient in Figure 9.2; i.e., $W_N^{-k} = (W_N^k)^*$.

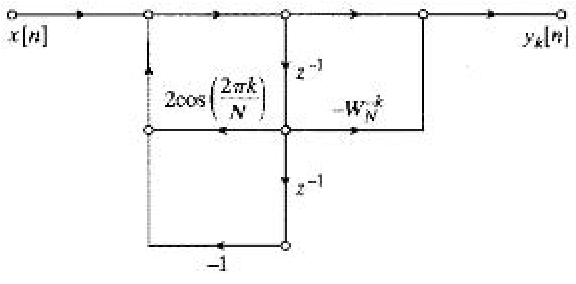


Figure P9.21-2

9.22. Consider the system shown in Figure P9.22. The subsystem from $x[n]$ to $y[n]$ is a causal, LTI system implementing the difference equation

$$y[n] = x[n] + ay[n-1].$$

$x[n]$ is a finite length sequence of length 90, i.e.,

$$x[n] = 0 \quad \text{for } n < 0 \text{ and } n > 89.$$

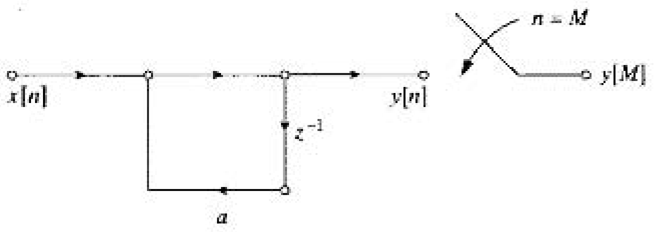


Figure P9.22

Determine a choice for the complex constant a and a choice for the sampling instant M so that

$$y[M] = X(e^{j\omega})\Big|_{\omega=2\pi/60}.$$

9.23. Construct a flow graph for a 16-point radix-2 decimation-in-time FFT algorithm. Label all multipliers in terms of powers of W_{16}, and also label any branch transmittances that are equal to -1. Label the input and output nodes with the appropriate values of the input and DFT sequences, respectively. Determine the number of real multiplications and the number of real additions required to implement the flow graph.

9.24. It is suggested that if you have an FFT subroutine for computing a length-N DFT, the inverse DFT of an N-point sequence $X[k]$ can be implemented using this subroutine as follows:

1. Swap the real and imaginary parts of each DFT coefficient $X[k]$.
2. Apply the FFT routine to this input sequence.
3. Swap the real and imaginary parts of the output sequence.
4. Scale the resulting sequence by $\frac{1}{N}$ to obtain the sequence $x[n]$, corresponding to the inverse DFT of $X[k]$.

Determine whether this procedure works as claimed. If it doesn't, propose a simple modification that will make it work.

9.25. The DFT is a sampled version of the DTFT of a finite-length sequence; i.e.,

$$\begin{aligned} X[k] &= X(e^{j(2\pi/N)k}) \\ &= X(e^{j\omega_k})\Big|_{\omega_k=(2\pi/N)k} \\ &= \sum_{n=0}^{N-1} x[n]e^{-j(2\pi/N)kn} \qquad k = 0, 1, \ldots, N-1. \end{aligned} \tag{P9.25-1}$$

Furthermore, an FFT algorithm is an efficient way to compute the values $X[k]$.

Now consider a finite-length sequence $x[n]$ whose length is N samples. We want to evaluate $X(z)$, the z-transform of the finite-length sequence, at the following points in the z-plane

$$z_k = re^{j(2\pi/N)k} \qquad k = 0, 1, \ldots, N-1,$$

where r is a positive number. We have available an FFT algorithm.

(a) Plot the points z_k in the z-plane for the case $N = 8$ and $r = 0.9$.

(b) Write an equation [similar to Eq. (P9.25-1) above] for $X(z_k)$ that shows that $X(z_k)$ is the DFT of a modified sequence $\tilde{x}[n]$. What is $\tilde{x}[n]$?

(c) Describe an algorithm for computing $X(z_k)$ using the given FFT function. (*Direct evaluation is not an option.*) You may describe your algorithm using any combination of English text and equations, but you must give a step-by-step procedure that starts with the sequence $x[n]$ and ends with $X(z_k)$.

9.26. We are given a finite-length sequence $x[n]$ of length 627 (i.e., $x[n] = 0$ for $n < 0$ and $n > 626$), and we have available a program that will compute the DFT of a sequence of any length $N = 2^\nu$.

For the given sequence, we want to compute samples of the DTFT at frequencies

$$\omega_k = \frac{2\pi}{627} + \frac{2\pi k}{256}, \qquad k = 0, 1, \ldots, 255.$$

Specify how to obtain a new sequence $y[n]$ from $x[n]$ such that the desired frequency samples can be obtained by applying the available FFT program to $y[n]$ with ν *as small as possible*.

9.27. A finite-length signal of length $L = 500$ ($x[n] = 0$ for $n < 0$ and $n > L - 1$) is obtained by sampling a continuous-time signal with sampling rate 10,000 samples per second. We wish to compute samples of the z-transform of $x[n]$ at the N equally spaced points $z_k = (0.8)e^{j2\pi k/N}$, for $0 \le k \le N-1$, with an effective frequency spacing of 50 Hz or less.

(a) Determine the minimum value for N if $N = 2^\nu$.

(b) Determine a sequence $y[n]$ of length N, where N is as determined in part (a), such that its DFT $Y[k]$ is equal to the desired samples of the z-transform of $x[n]$.

9.28. You are asked to build a system that computes the DFT of a 4-point sequence

$$x[0], x[1], x[2], x[3].$$

You can purchase any number of computational units at the per-unit cost shown in Table 9.1.

TABLE 9.1

Module	Per-Unit Cost
8-point DFT	\$1
8-point IDFT	\$1
adder	\$10
multiplier	\$100

Design a system of the lowest possible cost. Draw the associated block diagram and indicate the system cost.

Advanced Problems

9.29. Consider an N-point sequence $x[n]$ with DFT $X[k], k = 0, 1, \ldots, N-1$. The following algorithm computes the even-indexed DFT values $X[k], k = 0, 2, \ldots, N-2$, for N even, using only a single $N/2$-point DFT:

1. Form the sequence $y[n]$ by time aliasing, i.e.,

$$y[n] = \begin{cases} x[n] + x[n+N/2], & 0 \le n \le N/2 - 1, \\ 0, & \text{otherwise}. \end{cases}$$

2. Compute $Y[r], r = 0, 1, \ldots, (N/2) - 1$, the $N/2$-point DFT of $y[n]$.
3. Then the even-indexed values of $X[k]$ are $X[k] = Y[k/2]$, for $k = 0, 2, \ldots, N-2$.

(a) Show that the preceding algorithm produces the desired results.

(b) Now suppose that we form a finite-length sequence $y[n]$ from a sequence $x[n]$ by

$$y[n] = \begin{cases} \sum_{r=-\infty}^{\infty} x[n + rM], & 0 \le n \le M - 1, \\ 0, & \text{otherwise}. \end{cases}$$

Determine the relationship between the M-point DFT $Y[k]$ and $X(e^{j\omega})$, the Fourier transform of $x[n]$. Show that the result of part (a) is a special case of the result of part (b).

(c) Develop an algorithm similar to the one in part (a) to compute the odd-indexed DFT values $X[k], k = 1, 3, \ldots, N-1$, for N even, using only a single $N/2$-point DFT.

9.30. The system in Figure P9.30 computes an N-point (where N is an even number) DFT $X[k]$ of an N-point sequence $x[n]$ by decomposing $x[n]$ into two $N/2$-point sequences $g_1[n]$ and $g_2[n]$, computing the $N/2$-point DFT's $G_1[k]$ and $G_2[k]$, and then combining these to form $X[k]$.

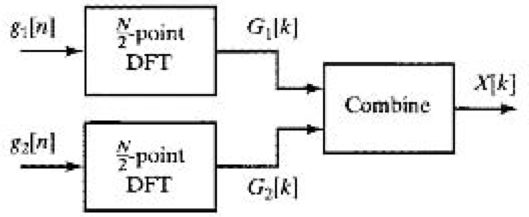


Figure P9.30

If $g_1[n]$ is the even-indexed values of $x[n]$ and $g_2[n]$ is the odd-indexed values of $x[n]$ i.e., $g_1[n] = x[2n]$ and $g_2[n] = x[2n+1]$ then $X[k]$ will be the DFT of $x[n]$.

In using the system in Figure P9.30 an error is made in forming $g_1[n]$ and $g_2[n]$, such that $g_1[n]$ is **incorrectly** chosen as the odd-indexed values and $g_2[n]$ as the even indexed values but $G_1[k]$ and $G_2[k]$ are still combined as in Figure P9.30 and the incorrect sequence $\hat{X}[k]$ results. Express $\hat{X}[k]$ in terms of $X[k]$.

9.31. In Section 9.3.2, it was asserted that the transpose of the flow graph of an FFT algorithm is also the flow graph of an FFT algorithm. The purpose of this problem is to develop that result for radix-2 FFT algorithms.

(a) The basic butterfly for the decimation-in-frequency radix-2 FFT algorithm is depicted in Figure P9.31-1. This flow graph represents the equations

$$X_m[p] = X_{m-1}[p] + X_{m-1}[q],$$
$$X_m[q] = (X_{m-1}[p] - X_{m-1}[q])W_N^r.$$

Starting with these equations, show that $X_{m-1}[p]$ and $X_{m-1}[q]$ can be computed from $X_m[p]$ and $X_m[q]$, respectively, using the butterfly shown in Figure P9.31-2.

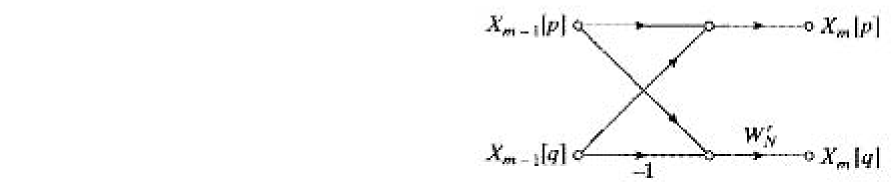


Figure P9.31-1

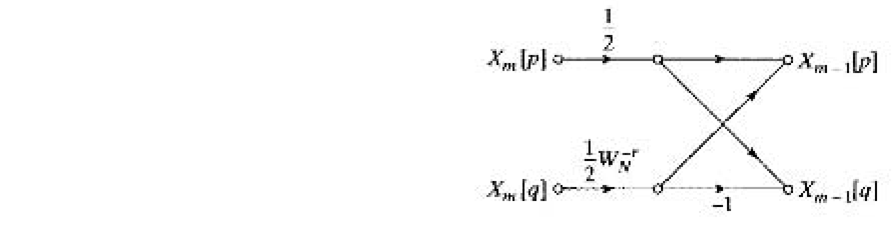


Figure P9.31-2

(b) In the decimation-in-frequency algorithm of Figure 9.22, $X_\mu[r]$, $r = 0, 1, \ldots, N-1$ is the DFT $X[k]$ arranged in bit-reversed order, and $X_0[r] = x[r]$, $r = 0, 1, \ldots, N-1$; i.e., the zero$^{\text{th}}$ array is the input sequence arranged in normal order. If each butterfly in Figure 9.22 is replaced by the appropriate butterfly of the form of Figure P9.31, the result would be a flow graph for computing the sequence $x[n]$ (in normal order) from the DFT $X[k]$ (in bit-reversed order). Draw the resulting flow graph for $N = 8$.

(c) The flow graph obtained in part (b) represents an *inverse* DFT algorithm, i.e., an algorithm for computing

$$x[n] = \frac{1}{N}\sum_{n=0}^{N-1} X[k]W_N^{-kn}, \qquad n = 0, 1, \ldots, N-1.$$

Modify the flow graph obtained in part (b) so that it computes the DFT

$$X[k] = \sum_{n=0}^{N-1} x[n]W_N^{kn}, \qquad k = 0, 1, \ldots, N-1,$$

rather than the inverse DFT.

(d) Observe that the result in part (c) is the transpose of the decimation-in-frequency algorithm of Figure 9.22 and that it is identical to the decimation-in-time algorithm depicted in Figure 9.11. Does it follow that, to each decimation-in-time algorithm (e.g., Figures 9.15–9.17), there corresponds a decimation-in-frequency algorithm that is the transpose of the decimation-in-time algorithm and vice versa? Explain.

9.32. We want to implement a 6-point decimation-in-time FFT using a mixed radix approach. One option is to first take three 2-point DFTs, and then use the results to compute the 6-point DFT. For this option:

(a) Draw a flowgraph to show what a 2-point DFT calculates. Also, fill in the parts of the flowgraph in Figure P9.32-1 involved in calculating the DFT values X_0, X_1, and X_4.

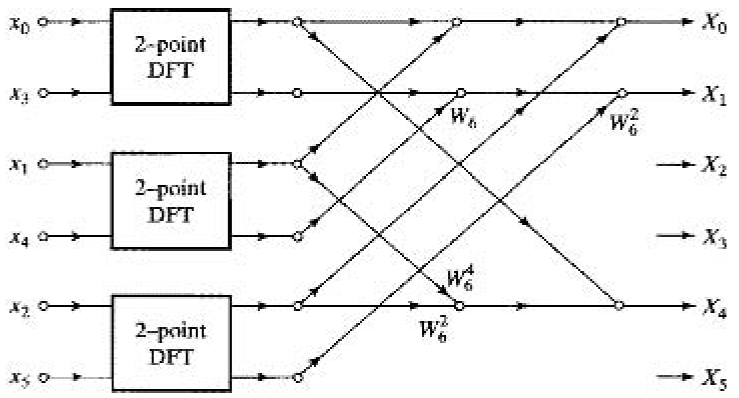


Figure P9.32-1

(b) How many complex multiplications does this option require? (Multiplying a number by -1 does not count as a complex multiplication.)

A second option is to start with two 3-point DFTs, and then use the results to compute the 6-point DFT.

(c) Draw a flowgraph to show what a 3-point DFT calculates. Also, fill in all of the flowgraph in Figure P9.32-2 and briefly explain how you derived your implementation.
(d) How many complex multiplications does this option require?

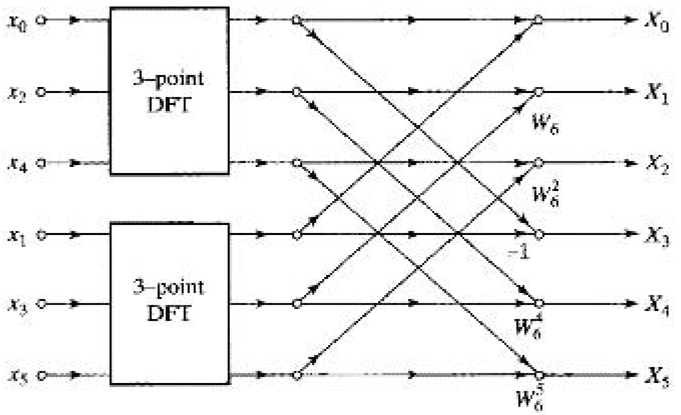


Figure P9.32-2

9.33. The decimation-in-frequency FFT algorithm was developed in Section 9.3 for radix 2, i.e., $N = 2^\nu$. A similar approach leads to a radix-3 algorithm when $N = 3^\nu$.

(a) Draw a flow graph for a 9-point decimation-in-frequency algorithm using a 3×3 decomposition of the DFT.
(b) For $N = 3^\nu$, how many complex multiplications by powers of W_N are needed to compute the DFT of an N-point complex sequence using a radix-3 decimation-in-frequency FFT algorithm?
(c) For $N = 3^\nu$, is it possible to use in-place computation for the radix-3 decimation-in-frequency algorithm?

9.34. We have seen that an FFT algorithm can be viewed as an interconnection of butterfly computational elements. For example, the butterfly for a radix-2 decimation-in-frequency FFT algorithm is shown in Figure P9.34-1. The butterfly takes two complex numbers as input and produces two complex numbers as output. Its implementation requires a complex multiplication by W_N^r, where r is an integer that depends on the location of the butterfly in the flow graph of the algorithm. Since the complex multiplier is of the form $W_N^r = e^{j\theta}$, the CORDIC (coordinate rotation digital computer) rotator algorithm (see Problem 9.46) can be used to implement the complex multiplication efficiently. Unfortunately, while the CORDIC rotator algorithm accomplishes the desired change of angle, it also introduces a fixed magnification that is independent of the angle θ. Thus, if the CORDIC rotator algorithm were used to implement the multiplications by W_N^r, the butterfly of Figure P9.34-1 would be replaced by the butterfly of Figure P9.34-2, where G represents the fixed magnification factor of the CORDIC rotator. (We assume no error in approximating the angle of rotation.) If each butterfly in the flow graph of the decimation-in-frequency FFT algorithm is replaced by the butterfly of Figure P9.34-2, we obtain a modified FFT algorithm for which the flow graph would be as shown in Figure P9.34-3 for $N = 8$. The output of this modified algorithm would not be the desired DFT.

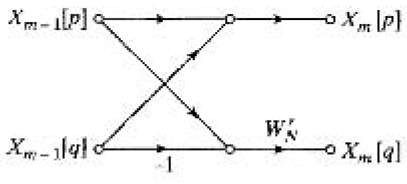


Figure P9.34-1

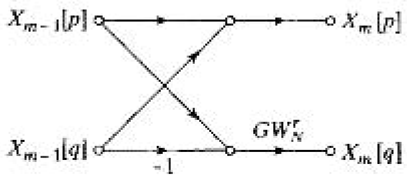


Figure P9.34-2

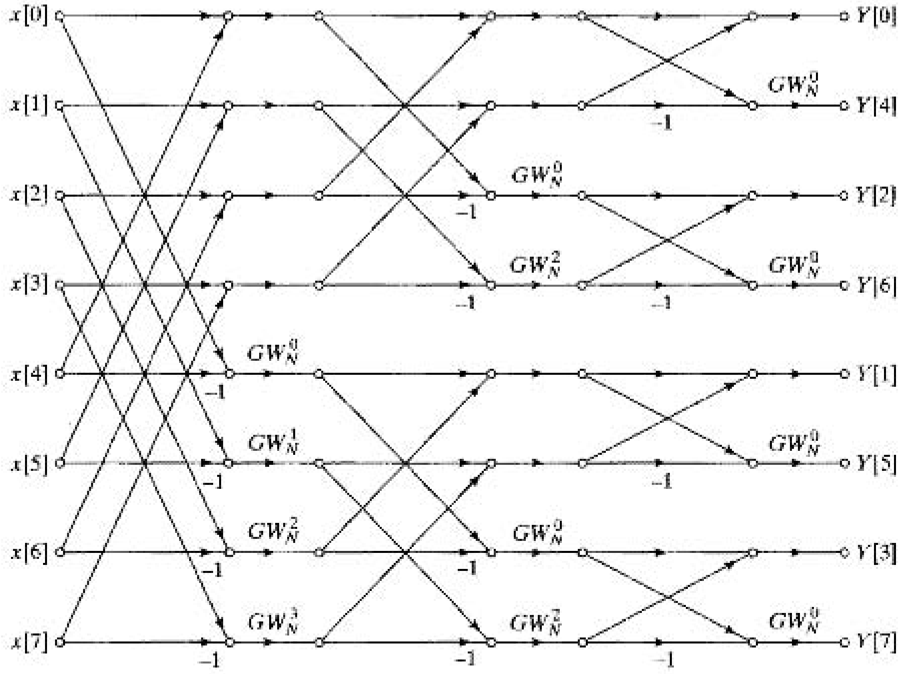


Figure P9.34-3

(a) Show that the output for the modified FFT algorithm is $Y[k] = W[k]X[k]$, where $X[k]$ is the correct DFT of the input sequence $x[n]$ and $W[k]$ is a function of G, N, and k.

(b) The sequence $W[k]$ can be described by a particularly simple rule. Find this rule and indicate its dependence on G, N, and k.

(c) Suppose that we wish to preprocess the input sequence $x[n]$ to compensate for the effect of the modified FFT algorithm. Determine a procedure for obtaining a sequence $\hat{x}[n]$ from $x[n]$ such that if $\hat{x}[n]$ is the input to the modified FFT algorithm, then the output will be $X[k]$, the correct DFT of the original sequence $x[n]$.

9.35. This problem deals with the efficient computation of samples of the z-transform of a finite-length sequence. Using the chirp transform algorithm, develop a procedure for computing values of $X(z)$ at 25 points spaced uniformly on an arc of a circle of radius 0.5, beginning at an angle of $-\pi/6$ and ending at an angle of $2\pi/3$. The length of the sequence is 100 samples.

9.36. Consider a 1024-point sequence $x[n]$ constructed by interleaving two 512-point sequences $x_e[n]$ and $x_o[n]$. Specifically,

$$x[n] = \begin{cases} x_e[n/2], \text{ if } n = 0, 2, 4, \ldots, 1022; \\ x_o[(n-1)/2], \text{ if } n = 1, 3, 5, \ldots, 1023; \\ 0, \text{ for } n \text{ outside of the range } 0 \le n \le 1023. \end{cases}$$

Let $X[k]$ denote the 1024-point DFT of $x[n]$ and $X_e[k]$ and $X_o[k]$ denote the 512-point DFTs of $x_e[n]$ and $x_o[n]$, respectively. Given $X[k]$ we would like to obtain $X_e[k]$ from $X[k]$ in a computationally efficient way where computational efficiency is measured in terms of the total number of complex multiplies and adds required. One not-very-efficient approach is as shown in Figure P9.36:

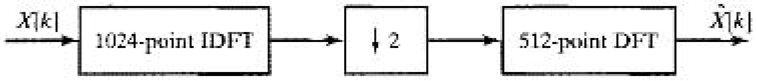


Figure P9.36

Specify the most efficient algorithm that you can (certainly more efficient than the block diagram of Figure P9.36) to obtain $X_e[k]$ from $X[k]$.

9.37. Suppose that a program is available that computes the DFT of a complex sequence. If we wish to compute the DFT of a real sequence, we may simply specify the imaginary part to be zero and use the program directly. However, the symmetry of the DFT of a real sequence can be used to reduce the amount of computation.

(a) Let $x[n]$ be a real-valued sequence of length N, and let $X[k]$ be its DFT with real and imaginary parts denoted $X_R[k]$ and $X_I[k]$, respectively; i.e.,

$$X[k] = X_R[k] + jX_I[k].$$

Show that if $x[n]$ is real, then $X_R[k] = X_R[N-k]$ and $X_I[k] = -X_I[N-k]$ for $k = 1, \ldots, N-1$.

(b) Now consider two real-valued sequences $x_1[n]$ and $x_2[n]$ with DFTs $X_1[k]$ and $X_2[k]$, respectively. Let $g[n]$ be the complex sequence $g[n] = x_1[n] + jx_2[n]$, with corresponding DFT $G[k] = G_R[k] + jG_I[k]$. Also, let $G_{\text{OR}}[k]$, $G_{\text{ER}}[k]$, $G_{\text{OI}}[k]$ and $G_{\text{EI}}[k]$ denote, respectively, the odd part of the real part, the even part of the real part, the odd part of the imaginary part, and the even part of the imaginary part of $G[k]$. Specifically, for $1 \le k \le N-1$,

$$G_{\text{OR}}[k] = \tfrac{1}{2}\{G_R[k] - G_R[N-k]\},$$
$$G_{\text{ER}}[k] = \tfrac{1}{2}\{G_R[k] + G_R[N-k]\},$$
$$G_{\text{OI}}[k] = \tfrac{1}{2}\{G_I[k] - G_I[N-k]\},$$
$$G_{\text{EI}}[k] = \tfrac{1}{2}\{G_I[k] + G_I[N-k]\},$$

and $G_{\text{OR}}[0] = G_{\text{OI}}[0] = 0$, $G_{\text{ER}}[0] = G_R[0]$, $G_{\text{EI}}[0] = G_I[0]$. Determine expressions for $X_1[k]$ and $X_2[k]$ in terms of $G_{\text{OR}}[k]$, $G_{\text{ER}}[k]$, $G_{\text{OI}}[k]$, and $G_{\text{EI}}[k]$.

(c) Assume that $N = 2^\nu$ and that a radix-2 FFT program is available to compute the DFT. Determine the number of real multiplications and the number of real additions required to compute both $X_1[k]$ and $X_2[k]$ by (i) using the program twice (with the imaginary part of the input set to zero) to compute the two complex N-point DFTs $X_1[k]$ and $X_2[k]$ separately and (ii) using the scheme suggested in part (b), which requires only one N-point DFT to be computed.

(d) Assume that we have only one real N-point sequence $x[n]$, where N is a power of 2. Let $x_1[n]$ and $x_2[n]$ be the two real $N/2$-point sequences $x_1[n] = x[2n]$ and $x_2[n] = x[2n+1]$, where $n = 0, 1, \ldots, (N/2) - 1$. Determine $X[k]$ in terms of the $(N/2)$-point DFTs $X_1[k]$ and $X_2[k]$.

(e) Using the results of parts (b), (c), and (d), describe a procedure for computing the DFT of the real N-point sequence $x[n]$ utilizing only one $N/2$-point FFT computation. Determine the numbers of real multiplications and real additions required by this procedure, and compare these numbers with the numbers required if the $X[k]$ is computed using one N-point FFT computation with the imaginary part set to zero.

9.38. Let $x[n]$ and $h[n]$ be two real finite-length sequences such that

$$x[n] = 0 \qquad \text{for } n \text{ outside the interval } 0 \le n \le L-1,$$
$$h[n] = 0 \qquad \text{for } n \text{ outside the interval } 0 \le n \le P-1.$$

We wish to compute the sequence $y[n] = x[n] * h[n]$, where $*$ denotes ordinary convolution.

(a) What is the length of the sequence $y[n]$?

(b) For direct evaluation of the convolution sum, how many real multiplications are required to compute all of the nonzero samples of $y[n]$? The following identity may be useful:

$$\sum_{k=1}^{N} k = \frac{N(N+1)}{2}.$$

(c) State a procedure for using the DFT to compute all of the nonzero samples of $y[n]$. Determine the minimum size of the DFTs and inverse DFTs in terms of L and P.

(d) Assume that $L = P = N/2$, where $N = 2^\nu$ is the size of the DFT. Determine a formula for the number of real multiplications required to compute all the nonzero values of $y[n]$ using the method of part (c) if the DFTs are computed using a radix-2 FFT algorithm. Use this formula to determine the minimum value of N for which the FFT method requires fewer real multiplications than the direct evaluation of the convolution sum.

9.39. In Section 8.7.3, we showed that linear time-invariant filtering can be implemented by sectioning the input signal into finite-length segments and using the DFT to implement circular convolutions on these segments. The two methods discussed were called the overlap–add and the overlap–save methods. If the DFTs are computed using an FFT algorithm, these sectioning methods can require fewer complex multiplications per output sample than the direct evaluation of the convolution sum.

(a) Assume that the complex input sequence $x[n]$ is of infinite duration and that the complex impulse response $h[n]$ is of length P samples, so that $h[n] \neq 0$ only for $0 \le n \le P-1$. Also, assume that the output is computed using the overlap–save method, with the DFTs of length $L = 2^\nu$, and suppose that these DFTs are computed using a radix-2 FFT algorithm. Determine an expression for the number of complex multiplications required per output sample as a function of ν and P.

(b) Suppose that the length of the impulse response is $P = 500$. By evaluating the formula obtained in part (a), plot the number of multiplications per output sample as a function of ν for the values of $\nu \leq 20$ such that the overlap–save method applies. For what value of ν is the number of multiplications minimal? Compare the number of complex multiplications per output sample for the overlap–save method using the FFT with the number of complex multiplications per output sample required for direct evaluation of the convolution sum.

(c) Show that for large FFT lengths, the number of complex multiplications per output sample is approximately ν. Thus, beyond a certain FFT length, the overlap–save method is less efficient than the direct method. If $P = 500$, for what value of ν will the direct method be more efficient?

(d) Assume that the FFT length is twice the length of the impulse response (i.e., $L = 2P$), and assume that $L = 2^\nu$. Using the formula obtained in part (a), determine the smallest value of P such that the overlap–save method using the FFT requires fewer complex multiplications than the direct convolution method.

9.40. $x[n]$ is a 1024-point sequence that is nonzero only for $0 \leq n \leq 1023$. Let $X[k]$ be the 1024-point DFT of $x[n]$. Given $X[k]$, we want to compute $x[n]$ in the ranges $0 \leq n \leq 3$ and $1020 \leq n \leq 1023$ using the system in Figure P9.40. Note that the input to the system is the sequence of DFT coefficients. By selecting $m_1[n]$, $m_2[n]$, and $h[n]$, show how the system can be used to compute the desired samples of $x[n]$. Note that the samples $y[n]$ for $0 \leq n \leq 7$ must contain the desired samples of $x[n]$.

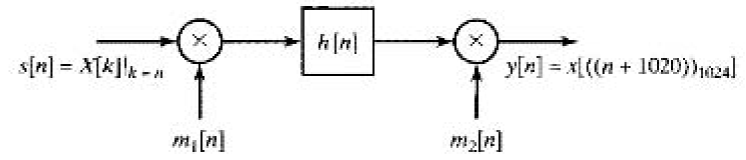


Figure P9.40

9.41. A system has been built for computing the 8-point DFT $Y[0], Y[1], \ldots, Y[7]$ of a sequence $y[0], y[1], \ldots, y[7]$. However, the system is not working properly: only the even DFT samples $Y[0], Y[2], Y[4], Y[6]$ are being computed correctly. To help you solve the problem, the data you can access are:

- the (correct) even DFT samples, $Y[0], Y[2], Y[4], Y[6]$;
- the first 4 input values $y[0], y[1], y[2], y[3]$ (the other inputs are unavailable).

(a) If $y[0] = 1$, and $y[1] = y[2] = y[3] = 0$, and $Y[0] = Y[2] = Y[4] = Y[6] = 2$, what are the missing values $Y[1], Y[3], Y[5], Y[7]$? Explain.

(b) You need to build an efficient system that computes the odd samples $Y[1], Y[3], Y[5], Y[7]$ for any set of inputs. The computational modules you have available are one 4-point DFT and one 4-point IDFT. Both are free. You can purchase adders, subtracters, or multipliers for \$10 each. Design a system of the lowest possible cost that takes as input

$$y[0], y[1], y[2], y[3], Y[0], Y[2], Y[4], Y[6]$$

and produces as output

$$Y[1], Y[3], Y[5], Y[7].$$

Draw the associated block diagram and indicate the total cost.

9.42. Consider a class of DFT-based algorithms for implementing a causal FIR filter with impulse response $h[n]$ that is zero outside the interval $0 \le n \le 63$. The input signal (for the FIR filter) $x[n]$ is segmented into an infinite number of possibly overlapping 128-point blocks $x_i[n]$, for i an integer and $-\infty \le i \le \infty$, such that

$$x_i[n] = \begin{cases} x[n], & iL \le n \le iL + 127, \\ 0, & \text{otherwise}, \end{cases}$$

where L is a positive integer.

Specify a method for computing

$$y_i[n] = x_i[n] * h[n]$$

for any i. Your answer should be in the form of a block diagram utilizing only the types of modules shown in Figures PP9.42-1 and PP9.42-2. A module may be used more than once or not at all.

The four modules in Figure P9.42-2 either use radix-2 FFTs to compute $X[k]$, the N-point DFT of $x[n]$, or use radix-2 inverse FFTs to compute $x[n]$ from $X[k]$.

Your specification must include the lengths of the FFTs and IFFTs used. For each "shift by n_0" module, you should also specify a value for n_0, the amount by which the input sequence is to be shifted.

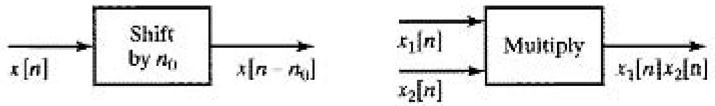
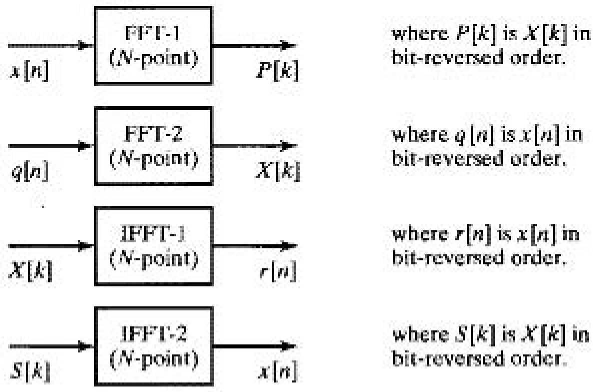


Figure P9.42-1



$x[n]$
FFT-1 (N-point)
$P[k]$
where $P[k]$ is $X[k]$ in bit-reversed order.
$q[n]$
FFT-2 (N-point)
$X[k]$
where $q[n]$ is $x[n]$ in bit-reversed order.
$X[k]$
IFFT-1 (N-point)
$r[n]$
where $r[n]$ is $x[n]$ in bit-reversed order.
$S[k]$
IFFT-2 (N-point)
$x[n]$
where $S[k]$ is $X[k]$ in bit-reversed order.

Figure P9.42-2

Extension Problems

9.43. In many applications (such as evaluating frequency responses and interpolation), it is of interest to compute the DFT of a short sequence that is "zero-padded." In such cases, a specialized "pruned" FFT algorithm can be used to increase the efficiency of computation (Markel, 1971). In this problem, we will consider pruning of the radix-2 decimation-in-frequency algorithm when the length of the input sequence is $M \le 2^\mu$ and the length of the DFT is $N = 2^\nu$, where $\mu < \nu$.

(a) Draw the complete flow graph of a decimation-in-frequency radix-2 FFT algorithm for $N = 16$. Label all branches appropriately.

(b) Assume that the input sequence is of length $M = 2$; i.e., $x[n] \neq 0$ only for $N = 0$ and $N = 1$. Draw a new flow graph for $N = 16$ that shows how the nonzero input samples propagate to the output DFT; i.e., eliminate or prune all branches in the flow graph of part (a) that represent operations on zero-inputs.

(c) In part (b), all of the butterflies in the first three stages of computation should have been effectively replaced by a half-butterfly of the form shown in Figure P9.43, and in the last stage, all the butterflies should have been of the regular form. For the general case where the length of the input sequence is $M \leq 2^{\mu}$ and the length of the DFT is $N = 2^{\nu}$, where $\mu < \nu$, determine the number of stages in which the pruned butterflies can be used. Also, determine the number of complex multiplications required to compute the N-point DFT of an M-point sequence using the pruned FFT algorithm. Express your answers in terms of ν and μ.

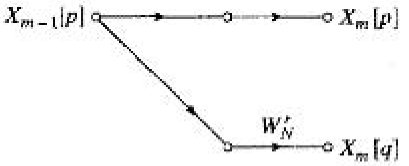


Figure P9.43

9.44. In Section 9.2, we showed that if N is divisible by 2, an N-point DFT may be expressed as

$$X[k] = G[((k))_{N/2}] + W_N^k H[((k))_{N/2}], \qquad 0 \leq k \leq N-1, \tag{P9.44-1}$$

where $G[k]$ is the $N/2$-point DFT of the sequence of even-indexed samples,

$$g[n] = x[2n], \qquad 0 \leq n \leq (N/2)-1,$$

and $H[k]$ is the $N/2$-point DFT of the odd-indexed samples,

$$h[n] = x[2n+1], \qquad 0 \leq n \leq (N/2)-1.$$

Note that $G[k]$ and $H[k]$ must be repeated periodically for $N/2 \leq k \leq N-1$ for Eq. (P9.44-1) to make sense. When $N = 2^{\nu}$, repeated application of this decomposition leads to the decimation-in-time FFT algorithm depicted for $N = 8$ in Figure 9.11. As we have seen, such algorithms require complex multiplications by the "twiddle" factors W_N^k. Rader and Brenner (1976) derived a new algorithm in which the multipliers are purely imaginary, thus requiring only two real multiplications and no real additions. In this algorithm, Eq. (P9.44-1) is replaced by the equations

$$X[0] = G[0] + F[0], \tag{P9.44-2}$$

$$X[N/2] = G[0] - F[0], \tag{P9.44-3}$$

$$X[k] = G[k] - \frac{1}{2} j \frac{F[k]}{\sin(2\pi k/N)}, \qquad k \neq 0, N/2. \tag{P9.44-4}$$

Here, $F[k]$ is the $N/2$-point DFT of the sequence

$$f[n] = x[2n+1] - x[2n-1] + Q,$$

where

$$Q = \frac{2}{N} \sum_{n=0}^{(N/2)-1} x[2n+1]$$

is a quantity that need be computed only once.

(a) Show that $F[0] = H[0]$ and therefore that Eqs. (P9.44-2) and (P9.44-3) give the same result as Eq. (P9.44-1) for $k = 0, N/2$.

(b) Show that

$$F[k] = H[k]W_N^k(W_N^{-k} - W_N^k)$$

for $k = 1, 2, \ldots, (N/2) - 1$. Use this result to obtain Eq. (P9.44-4). Why must we compute $X[0]$ and $X[N/2]$ using separate equations?

(c) When $N = 2^\nu$, we can apply Eqs. (P9.44-2)–(P9.44-4) repeatedly to obtain a complete decimation-in-time FFT algorithm. Determine formulas for the number of real multiplications and for the number of real additions as a function of N. In counting operations due to Eq. (P9.44-4), take advantage of any symmetries and periodicities, but do not exclude "trivial" multiplications by $\pm j/2$.

(d) Rader and Brenner (1976) state that FFT algorithms based on Eqs. (P9.44-2)–(P9.44-4) have "poor noise properties." Explain why this might be true.

9.45. A modified FFT algorithm called the *split-radix* FFT, or SRFFT, was proposed by Duhamel and Hollman (1984) and Duhamel (1986). The flow graph for the split-radix algorithm is similar to the radix-2 flow graph, but it requires fewer real multiplications. In this problem, we illustrate the principles of the SRFFT for computing the DFT $X[k]$ of a sequence $x[n]$ of length N.

(a) Show that the even-indexed terms of $X[k]$ can be expressed as the $N/2$-point DFT

$$X[2k] = \sum_{n=0}^{(N/2)-1} (x[n] + x[n + N/2])W_N^{2kn}$$

for $k = 0, 1, \ldots, (N/2) - 1$.

(b) Show that the odd-indexed terms of the DFT $X[k]$ can be expressed as the $N/4$-point DFTs

$$X[4k+1] = \sum_{n=0}^{(N/4)-1} \{(x[n] - x[n+N/2]) - j(x[n+N/4] - x[n+3N/4])\}W_N^n W_N^{4kn}$$

for $k = 0, 1, \ldots, (N/4) - 1$, and

$$X[4k+3] = \sum_{n=0}^{(N/4)-1} \{(x[n] - x[n+N/2]) + j(x[n+N/4] - x[n+3N/4])\}W_N^{3n} W_N^{4kn}$$

for $k = 0, 1, \ldots, (N/4) - 1$.

(c) The flow graph in Figure P9.45 represents the preceding decomposition of the DFT for a 16-point transform. Redraw this flow graph, labeling each branch with the appropriate multiplier coefficient.

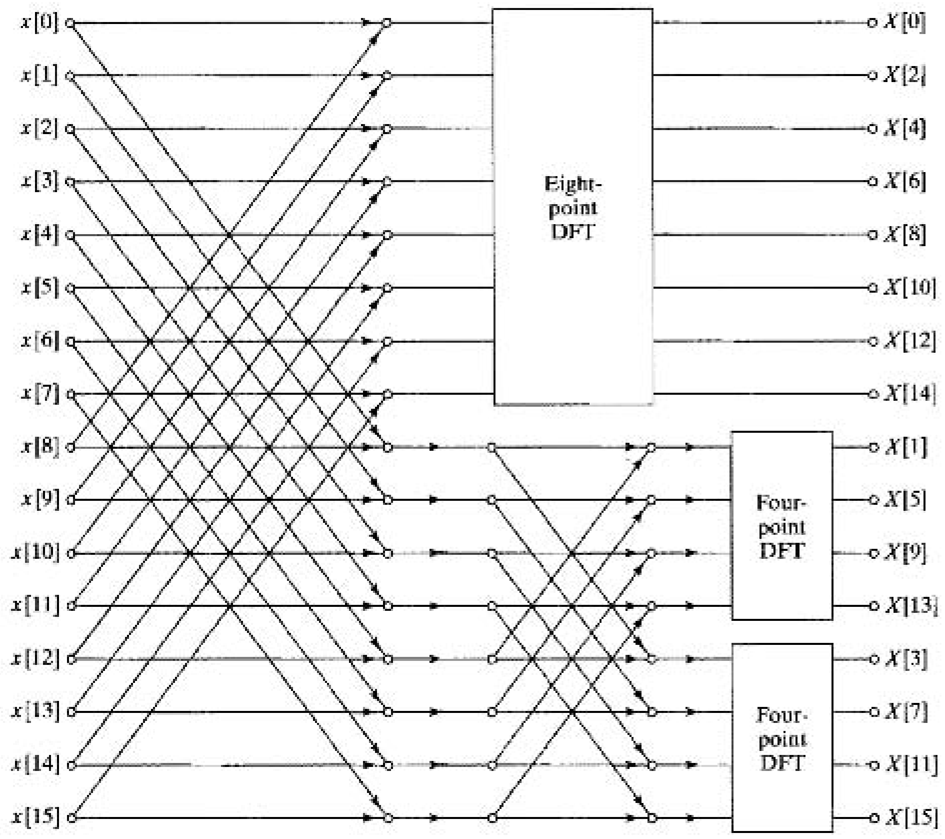


Figure P9.45

(d) Determine the number of real multiplications required to implement the 16-point transform when the SRFFT principle is applied to compute the other DFTs in Figure P9.45. Compare this number with the number of real multiplications required to implement a 16-point radix-2 decimation-in-frequency algorithm. In both cases, assume that multiplications by W_N^0 are not done.

9.46. In computing the DFT, it is necessary to multiply a complex number by another complex number whose magnitude is unity, i.e., $(X + jY)e^{j\theta}$. Clearly, such a complex multiplication changes only the angle of the complex number, leaving the magnitude unchanged. For this reason, multiplications by a complex number $e^{j\theta}$ are sometimes called *rotations*. In DFT or FFT algorithms, many different angles θ may be needed. However, it may be undesirable to store a table of all required values of $\sin\theta$ and $\cos\theta$, and computing these functions by a power series requires many multiplications and additions. With the CORDIC algorithm given by Volder (1959), the product $(X + jY)e^{j\theta}$ can be evaluated efficiently by a combination of additions, binary shifts, and table lookups from a small table.

(a) Define $\theta_i = \arctan(2^{-i})$. Show that any angle $0 < \theta < \pi/2$ can be represented as

$$\theta = \sum_{i=0}^{M-1} \alpha_i\theta_i + \epsilon = \hat{\theta} + \epsilon,$$

where $\alpha_i = \pm 1$ and the error ϵ is bounded by

$$|\epsilon| \leq \arctan(2^{-M}).$$

(b) The angles θ_i may be computed in advance and stored in a small table of length M. State an algorithm for obtaining the sequence $\{\alpha_i\}$ for $i = 0, 1, \ldots, M-1$, such that $\alpha_i = \pm 1$. Use your algorithm to determine the sequence $\{\alpha_i\}$ for representing the angle $\theta = 100\pi/512$ when $M = 11$.

(c) Using the result of part (a), show that the recursion

$$\begin{aligned} X_0 &= X, \\ Y_0 &= Y, \\ X_i &= X_{i-1} - \alpha_{i-1}Y_{i-1}2^{-i+1}, \qquad i = 1, 2, \ldots, M, \\ Y_i &= Y_{i-1} + \alpha_{i-1}X_{i-1}2^{-i+1}, \qquad i = 1, 2, \ldots, M, \end{aligned}$$

will produce the complex number

$$(X_M + jY_M) = (X + jY)G_M e^{j\hat{\theta}},$$

where $\hat{\theta} = \sum_{i=0}^{M-1} \alpha_i\theta_i$ and G_M is real, is positive, and does not depend on θ. That is, the original complex number is rotated in the complex plane by an angle $\hat{\theta}$ and magnified by the constant G_M.

(d) Determine the magnification constant G_M as a function of M.

9.47. In Section 9.3, we developed the decimation-in-frequency FFT algorithm for radix 2, i.e., $N = 2^\nu$. It is possible to formulate a similar algorithm for the general case of $N = m^\nu$, where m is an integer. Such an algorithm is known as a radix-m FFT algorithm. In this problem, we will examine the radix-3 decimation-in-frequency FFT for the case when $N = 9$, i.e., the input sequence $x[n] = 0$ for $n < 0$ and $n > 8$.

(a) Formulate a method of computing the DFT samples $X[3k]$ for $k = 0, 1, 2$. Consider defining $X_1[k] = X(e^{j\omega_k})|_{\omega_k = 2\pi k/3}$. How can you define a time sequence $x_1[n]$ in terms of $x[n]$ such that the 3-point DFT of $x_1[n]$ is $X_1[k] = X[3k]$?

(b) Now define a sequence $x_2[n]$ in terms of $x[n]$ such that the 3-point DFT of $x_2[n]$ is $X_2[k] = X[3k+1]$ for $k = 0, 1, 2$. Similarly, define $x_3[n]$ such that its 3-point DFT $X_3[k] = X[3k+2]$ for $k = 0, 1, 2$. Note that we have now defined the 9-point DFT as three 3-point DFTs from appropriately constructed 3-point sequences.

(c) Draw the signal flow graph for the $N = 3$ DFT, i.e., the radix-3 butterfly.

(d) Using the results for parts (a) and (b), sketch the signal flow graph for the system that constructs the sequences $x_1[n]$, $x_2[n]$, and $x_3[n]$, and then use 3-point DFT boxes on these sequences to produce $X[k]$ for $k = 0, \ldots, 8$. Note that in the interest of clarity, you should not draw the signal flow graph for the $N = 3$ DFTs, but simply use boxes labeled "$N = 3$ DFT." The interior of these boxes is the system you drew for part (c).

(e) Appropriate factoring of the powers of W_9 in the system you drew in part (d) allows these systems to be drawn as $N = 3$ DFTs, followed by "twiddle" factors analogous to those in the radix-2 algorithm. Redraw the system in part (d) such that it consists entirely of $N = 3$ DFTs with "twiddle" factors. This is the complete formulation of the radix-3 decimation-in-frequency FFT for $N = 9$.

(f) How many complex multiplications are required to compute a 9-point DFT using a direct implementation of the DFT equation? Contrast this with the number of complex multiplications required by the system you drew in part (e). In general, how many complex multiplications are required for the radix-3 FFT of a sequence of length $N = 3^\nu$?

9.48. Bluestein (1970) showed that if $N = M^2$, then the chirp transform algorithm has a recursive implementation.

(a) Show that the DFT can be expressed as the convolution

$$X[k] = h^*[k] \sum_{n=0}^{N-1} (x[n]h^*[n])h[k-n],$$

where * denotes complex conjugation and

$$h[n] = e^{j(\pi/N)n^2}, \qquad -\infty < n < \infty.$$

(b) Show that the desired values of $X[k]$ (i.e., for $k = 0, 1, \ldots, N-1$) can also be obtained by evaluating the convolution of part (a) for $k = N, N+1, \ldots, 2N-1$.

(c) Use the result of part (b) to show that $X[k]$ is also equal to the output of the system shown in Figure P9.48 for $k = N, N+1, \ldots, 2N-1$, where $\hat{h}[k]$ is the finite-duration sequence

$$\hat{h}[k] = \begin{cases} e^{j(\pi/N)k^2}, & 0 \le k \le 2N-1, \\ 0, & \text{otherwise.} \end{cases}$$

(d) Using the fact that $N = M^2$, show that the system function corresponding to the impulse response $\hat{h}[k]$ is

$$\hat{H}(z) = \sum_{k=0}^{2N-1} e^{j(\pi/N)k^2} z^{-k}$$
$$= \sum_{r=0}^{M-1} e^{j(\pi/N)r^2} z^{-r} \frac{1 - z^{-2M^2}}{1 + e^{j(2\pi/M)r} z^{-M}}.$$

Hint: Express k as $k = r + \ell M$.

(e) The expression for $\hat{H}(z)$ obtained in part (d) suggests a recursive realization of the FIR system. Draw the flow graph of such an implementation.

(f) Use the result of part (e) to determine the total numbers of complex multiplications and additions required to compute all of the N desired values of $X[k]$. Compare those numbers with the numbers required for direct computation of $X[k]$.

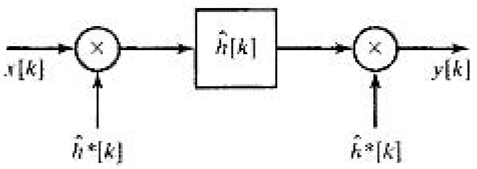


Figure P9.48

9.49. In the Goertzel algorithm for computation of the discrete Fourier transform, $X[k]$ is computed as

$$X[k] = y_k[N],$$

where $y_k[n]$ is the output of the network shown in Figure P9.49. Consider the implementation of the Goertzel algorithm using fixed-point arithmetic with rounding. Assume that the register length is B bits plus the sign, and assume that the products are rounded before additions. Also, assume that round-off noise sources are independent.

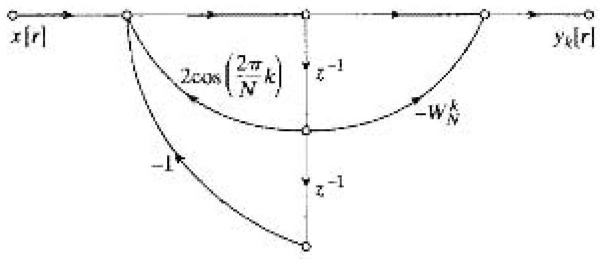


Figure P9.49

(a) Assuming that $x[n]$ is real, draw a flow graph of the linear-noise model for the finite-precision computation of the real and imaginary parts of $X[k]$. Assume that multiplication by ± 1 produces no round-off noise.

(b) Compute the variance of the round-off noise in both the real part and the imaginary part of $X[k]$.

9.50. Consider direct computation of the DFT using fixed-point arithmetic with rounding. Assume that the register length is B bits plus the sign (i.e., a total of $B+1$ bits) and that the round-off noise introduced by any real multiplication is independent of that produced by any other real multiplication. Assuming that $x[n]$ is real, determine the variance of the round-off noise in both the real part and the imaginary part of each DFT value $X[k]$.

9.51. In implementing a decimation-in-time FFT algorithm, the basic butterfly computation is

$$X_m[p] = X_{m-1}[p] + W_N^r X_{m-1}[q],$$
$$X_m[q] = X_{m-1}[p] - W_N^r X_{m-1}[q].$$

In using fixed-point arithmetic to implement the computations, it is commonly assumed that all numbers are scaled to be less than unity. Therefore, to avoid overflow, it is necessary to ensure that the real numbers that result from the butterfly computations do not exceed unity.

(a) Show that if we require

$$|X_{m-1}[p]| < \tfrac{1}{2} \quad \text{and} \quad |X_{m-1}[q]| < \tfrac{1}{2},$$

then overflow cannot occur in the butterfly computation; i.e.,

$$|\mathcal{R}e\{X_m[p]\}| < 1, \qquad |\mathcal{I}m\{X_m[p]\}| < 1,$$

and

$$|\mathcal{R}e\{X_m[q]\}| < 1, \qquad |\mathcal{I}m\{X_m[q]\}| < 1.$$

(b) In practice, it is easier and most convenient to require

$$|\mathcal{R}e\{X_{m-1}[p]\}| < \tfrac{1}{2}, \qquad |\mathcal{I}m\{X_{m-1}[p]\}| < \tfrac{1}{2},$$

and

$$|\mathcal{R}e\{X_{m-1}[q]\}| < \tfrac{1}{2}, \qquad |\mathcal{I}m\{X_{m-1}[q]\}| < \tfrac{1}{2}.$$

Are these conditions sufficient to guarantee that overflow cannot occur in the decimation-in-time butterfly computation? Explain.

9.52. In deriving formulas for the noise-to-signal ratio for the fixed-point radix-2 decimation-in-time FFT algorithm, we assumed that each output node was connected to $(N-1)$ butterfly computations, each of which contributed an amount $\sigma_B^2 = \frac{1}{3}\cdot 2^{-2B}$ to the output noise variance. However, when $W_N^r = \pm 1$ or $\pm j$, the multiplications can in fact be done without error. Thus, if the results derived in Section 9.7 are modified to account for this fact, we obtain a less pessimistic estimate of quantization noise effects.

(a) For the decimation-in-time algorithm discussed in Section 9.7, determine, for each stage, the number of butterflies that involve multiplication by either ± 1 or $\pm j$.

(b) Use the result of part (a) to find improved estimates of the output noise variance, Eq. (9.58), and noise-to-signal ratio, Eq. (9.68), for odd values of k. Discuss how these estimates are different for even values of k. Do not attempt to find a closed form expression of these quantities for even values of k.

(c) Repeat parts (a) and (b) for the case where the output of each stage is attenuated by a factor of $\frac{1}{2}$; i.e., derive modified expressions corresponding to Eq. (9.71) for the output noise variance and Eq. (9.72) for the output noise-to-signal ratio, assuming that multiplications by ± 1 and $\pm j$ do not introduce error.

9.53. In Section 9.7 we considered a noise analysis of the decimation-in-time FFT algorithm of Figure 9.11. Carry out a similar analysis for the decimation-in-frequency algorithm of Figure 9.22, obtaining equations for the output noise variance and noise-to-signal ratio for scaling at the input and also for scaling by $\frac{1}{2}$ at each stage of computation.

9.54. In this problem, we consider a procedure for computing the DFT of four real symmetric or antisymmetric N-point sequences using only one N-point DFT computation. Since we are considering only finite-length sequences, by *symmetric* and *antisymmetric*, we explicitly mean *periodic symmetric* and *periodic antisymmetric*, as defined in Section 8.6.4. Let $x_1[n]$, $x_2[n]$, $x_3[n]$, and $x_4[n]$ denote the four real sequences of length N, and let $X_1[k]$, $X_2[k]$, $X_3[k]$, and $X_4[k]$ denote the corresponding DFTs. We assume first that $x_1[n]$ and $x_2[n]$ are symmetric and $x_3[n]$ and $x_4[n]$ are antisymmetric; i.e.,

$$x_1[n] = x_1[N-n], \qquad x_2[n] = x_2[N-n],$$
$$x_3[n] = -x_3[N-n], \qquad x_4[n] = -x_4[N-n],$$

for $n = 1, 2, \ldots, N-1$ and $x_3[0] = x_4[0] = 0$.

(a) Define $y_1[n] = x_1[n] + x_3[n]$ and let $Y_1[k]$ denote the DFT of $y_1[n]$. Determine how $X_1[k]$ and $X_2[k]$ can be recovered from $Y_1[k]$.

(b) $y_1[n]$ as defined in part (a) is a real sequence with symmetric part $x_1[n]$ and antisymmetric part $x_3[n]$. Similarly, we define the real sequence $y_2[n] = x_2[n] + x_4[n]$, and we let $y_3[n]$ be the complex sequence

$$y_3[n] = y_1[n] + jy_2[n].$$

First, determine how $Y_1[k]$ and $Y_2[k]$ can be determined from $Y_3[k]$, and then, using the results of part (a), show how to obtain $X_1[k]$, $X_2[k]$, $X_3[k]$, and $X_4[k]$ from $Y_3[k]$.

The result of part (b) shows that we can compute the DFT of four real sequences simultaneously with only one N-point DFT computation if two sequences are symmetric and the other two are antisymmetric. Now consider the case when all four are symmetric; i.e.,

$$x_i[n] = x_i[N-n], \qquad i = 1, 2, 3, 4,$$

for $n = 0, 1, \ldots, N-1$. For parts (c)–(f), assume $x_3[n]$ and $x_4[n]$ are real and symmetric, not antisymmetric.

(c) Consider a real symmetric sequence $x_3[n]$. Show that the sequence

$$u_3[n] = x_3[((n+1))_N] - x_3[((n-1))_N]$$

is an antisymmetric sequence; i.e., $u_3[n] = -u_3[N-n]$ for $n = 1, 2, \ldots, N-1$ and $u_3[0] = 0$.

(d) Let $U_3[k]$ denote the N-point DFT of $u_3[n]$. Determine an expression for $U_3[k]$ in terms of $X_3[k]$.

(e) By using the procedure of part (c), we can form the real sequence $y_1[n] = x_1[n] + u_3[n]$, where $x_1[n]$ is the symmetric part and $u_3[n]$ is the antisymmetric part of $y_1[n]$. Determine how $X_1[k]$ and $X_3[k]$ can be recovered from $Y_1[k]$.

(f) Now let $y_3[n] = y_1[n] + jy_2[n]$, where

$$y_1[n] = x_1[n] + u_3[n], \qquad y_2[n] = x_2[n] + u_4[n],$$

with

$$u_3[n] = x_3[((n+1))_N] - x_3[((n-1))_N],$$
$$u_4[n] = x_4[((n+1))_N] - x_4[((n-1))_N],$$

for $n = 0, 1, \ldots, N-1$. Determine how to obtain $X_1[k]$, $X_2[k]$, $X_3[k]$, and $X_4[k]$ from $Y_3[k]$. (Note that $X_3[0]$ and $X_4[0]$ cannot be recovered from $Y_3[k]$, and if N is even, $X_3[N/2]$ and $X_4[N/2]$ also cannot be recovered from $Y_3[k]$.)

9.55. The input and output of a linear time-invariant system satisfy a difference equation of the form

$$y[n] = \sum_{k=1}^{N} a_k y[n-k] + \sum_{k=0}^{M} b_k x[n-k].$$

Assume that an FFT program is available for computing the DFT of any finite-length sequence of length $L = 2^\nu$. Describe a procedure that utilizes the available FFT program to compute

$$H(e^{j(2\pi/512)k}) \qquad \text{for } k = 0, 1, \ldots, 511,$$

where $H(z)$ is the system function of the system.

9.56. Suppose that we wish to multiply two very large numbers (possibly thousands of bits long) on a 16-bit computer. In this problem, we will investigate a technique for doing this using FFTs.

(a) Let $p(x)$ and $q(x)$ be the two polynomials

$$p(x) = \sum_{i=0}^{L-1} a_i x^i, \qquad q(x) = \sum_{i=0}^{M-1} b_i x^i.$$

Show that the coefficients of the polynomial $r(x) = p(x)q(x)$ can be computed using circular convolution.

(b) Show how to compute the coefficients of $r(x)$ using a radix-2 FFT program. For what orders of magnitude of $(L + M)$ is this procedure more efficient than direct computation? Assume that $L + M = 2^\nu$ for some integer ν.

(c) Now suppose that we wish to compute the product of two very long positive binary integers u and v. Show that their product can be computed using polynomial multiplication, and describe an algorithm for computing the product using an FFT algorithm. If u is an 8000-bit number and v is a 1000-bit number, approximately how many real multiplications and additions are required to compute the product $u \cdot v$ using this method?

(d) Give a qualitative discussion of the effect of finite-precision arithmetic in implementing the algorithm of part (c).

9.57. The discrete Hartley transform (DHT) of a sequence $x[n]$ of length N is defined as

$$X_H[k] = \sum_{n=0}^{N-1} x[n]H_N[nk], \qquad k = 0, 1, \ldots, N-1,$$

where

$$H_N[a] = C_N[a] + S_N[a],$$

with

$$C_N[a] = \cos(2\pi a/N), \qquad S_N[a] = \sin(2\pi a/N).$$

Problem 8.68 explores the properties of the discrete Hartley transform in detail, particularly its circular convolution property.

(a) Verify that $H_N[a] = H_N[a+N]$, and verify the following useful property of $H_N[a]$:

$$\begin{aligned} H_N[a+b] &= H_N[a]C_N[b] + H_N[-a]S_N[b] \\ &= H_N[b]C_N[a] + H_N[-b]S_N[a]. \end{aligned}$$

(b) By decomposing $x[n]$ into its even-numbered points and odd-numbered points, and by using the identity derived in part (a), derive a fast DHT algorithm based on the decimation-in-time principle.

9.58. In this problem, we will write the FFT as a sequence of matrix operations. Consider the 8-point decimation-in-time FFT algorithm shown in Figure P9.58. Let a and f denote the input and output vectors, respectively. Assume that the input is in bit-reversed order and that the output is in normal order (compare with Figure 9.11). Let b, c, d, and e denote the intermediate vectors shown on the flow graph.

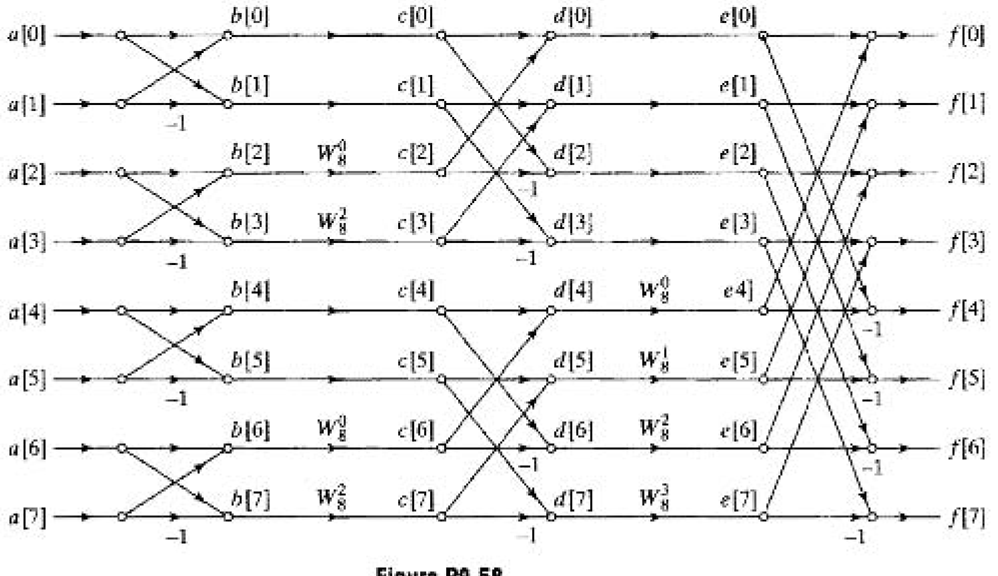


Figure P9.58

(a) Determine the matrices F_1, T_1, F_2, T_2, and F_3 such that

$$b = F_1 a,$$

$$c = T_1 b,$$

$$d = F_2 c,$$

$$e = T_2 d,$$

$$f = F_3 e.$$

(b) The overall FFT, taking input a and yielding output f can be described in matrix notation as $f = Qa$, where

$$Q = F_3 T_2 F_2 T_1 F_1.$$

Let Q^H be the complex (Hermitian) transpose of the matrix Q. Draw the flow graph for the sequence of operations described by Q^H. What does this structure compute?

(c) Determine $(1/N)Q^H Q$.

9.59. In many applications, there is a need to convolve long sequences $x[n]$ and $h[n]$ whose samples are integers. Since the sequences have integer coefficients, the result of the convolution $y[n] = x[n] * h[n]$ will naturally also have integer coefficients as well.

A major drawback of computing the convolution of integer sequences with FFTs is that floating-point arithmetic chips are more expensive than integer arithmetic chips. Also, rounding noise introduced during a floating-point computation may corrupt the result. In this problem, we consider a class of FFT algorithms known as *number-theoretic transforms* (NTTs), which overcome these drawbacks.

(a) Let $x[n]$ and $h[n]$ be N-point sequences and denote their DFTs by $X[k]$ and $H[k]$, respectively. Derive the circular convolution property of the DFT. Specifically, show that $Y[k] = X[k]H[k]$, where $y[n]$ is the N-point circular convolution of $x[n]$ and $h[n]$. Show that the circular convolution property holds as long as W_N in the DFT satisfies

$$\sum_{n=0}^{N-1} W_N^{nk} = \begin{cases} N, & k = 0, \\ 0, & k \neq 0. \end{cases} \qquad \text{(P9.59-1)}$$

The key to defining NTTs is to find an integer-valued W_N that satisfies Eq. (P9.59-1). This will enforce the orthogonality of the basis vectors required for the DFT to function properly. Unfortunately, no integer-valued W_N exists that has this property for standard integer arithmetic.

In order to overcome this problem, NTTs use integer arithmetic defined modulo some integer P. Throughout the current problem, we will assume that $P = 17$. That is, addition and multiplication are defined as standard integer addition and multiplication, followed by modulo $P = 17$ reduction. For example, $((23+18))_{17} = 7$, $((10+7))_{17} = 0$, $((23 \times 18))_{17} = 6$, and $((10 \times 7))_{17} = 2$. (Just compute the sum or product in the normal way, and then take the remainder modulo 17.)

(b) Let $P = 17$, $N = 4$, and $W_N = 4$. Verify that

$$\left(\left(\sum_{n=0}^{N-1} W_N^{nk}\right)\right)_P = \begin{cases} N, & k = 0, \\ 0, & k \neq 0. \end{cases}$$

(c) Let $x[n]$ and $h[n]$ be the sequences

$$x[n] = \delta[n] + 2\delta[n-1] + 3\delta[n-2],$$

$$h[n] = 3\delta[n] + \delta[n-1].$$

Compute the 4-point NTT $X[k]$ of $x[n]$ as follows:

$$X[k] = \left(\left(\sum_{n=0}^{N-1} x[n] W_N^{nk}\right)\right)_P.$$

Compute $H[k]$ in a similar fashion. Also, compute $Y[k] = H[k]X[k]$. Assume the values of P, N, and W_N given in part (a). *Be sure to use modulo 17 arithmetic for each operation throughout the computation, not just for the final result!*

(d) The inverse NTT of $Y[k]$ is defined by the equation

$$y[n] = \left(\left((N)^{-1} \sum_{k=0}^{N-1} Y[k] W_N^{-nk}\right)\right)_P. \quad \text{(P9.59-2)}$$

In order to compute this quantity properly, we must determine the *integers* $(1/N)^{-1}$ and W_N^{-1} such that

$$\left(\left((N)^{-1}N\right)\right)_P = 1,$$

$$\left(\left(W_N W_N^{-1}\right)\right)_P = 1.$$

Use the values of P, N, and W_N given in part (a), and determine the aforesaid integers.

(e) Compute the inverse NTT shown in Eq. (P9.59-2) using the values of $(N)^{-1}$ and W_N^{-1} determined in part (d). Check your result by manually computing the convolution $y[n] = x[n] * h[n]$.

9.60. Sections 9.2 and 9.3 focus on the fast Fourier transform for sequences where N is a power of 2. However, it is also possible to find efficient algorithms to compute the DFT when the length N has more than one prime factor, i.e., cannot be expressed as $N = m^\nu$ for some integer m. In this problem, you will examine the case where $N = 6$. The techniques described extend easily to other composite numbers. Burrus and Parks (1985) discuss such algorithms in more detail.

(a) The key to decomposing the FFT for $N = 6$ is to use the concept of an *index map*, proposed by Cooley and Tukey (1965) in their original paper on the FFT. Specifically, for the case of $N = 6$, we will represent the indices n and k as

$$n = 3n_1 + n_2 \quad \text{for } n_1 = 0, 1;\ n_2 = 0, 1, 2; \quad \text{(P9.60-1)}$$

$$k = k_1 + 2k_2 \quad \text{for } k_1 = 0, 1;\ k_2 = 0, 1, 2; \quad \text{(P9.60-2)}$$

Verify that using each possible value of n_1 and n_2 produces each value of $n = 0, \ldots, 5$ once and only once. Demonstrate that the same holds for k with each choice of k_1 and k_2.

(b) Substitute Eqs. (P9.60-1) and (P9.60-2) into the definition of the DFT to get a new expression for the DFT in terms of n_1, n_2, k_1, and k_2. The resulting equation should have a double summation over n_1 and n_2 instead of a single summation over n.

(c) Examine the W_6 terms in your equation carefully. You can rewrite some of these as equivalent expressions in W_2 and W_3.

(d) Based on part (c), group the terms in your DFT such that the n_2 summation is outside and the n_1 summation is inside. You should be able to write this expression so that it can be interpreted as three DFTs with $N = 2$, followed by some "twiddle" factors (powers of W_6), followed by two $N = 3$ DFTs.

(e) Draw the signal flow graph implementing your expression from part (d). How many complex multiplications does this require? How does this compare with the number of complex multiplications required by a direct implementation of the DFT equation for $N = 6$?

(f) Find an alternative indexing similar to Eqs. (P9.60-1) and (P9.60-2) that results in a signal flow graph that is two $N = 3$ DFTs followed by three $N = 2$ DFTs.