The two algorithms given above have the same complexity. However, they differ in the arrangement of the computations. In the following sections we exploit the divide-and-conquer approach to derive fast algorithms when the size of the DFT is restricted to be a power of 2 or a power of 4.

8.1.3 Radix-2 FFT Algorithms

In the preceding section we described four algorithms for efficient computation of the DFT based on the divide-and-conquer approach. Such an approach is applicable when the number N of data points is not a prime. In particular, the approach is very efficient when N is highly composite, that is, when N can be factored as $N = r_1 r_2 r_3 \cdots r_\nu$, where the $\{r_j\}$ are prime.

Of particular importance is the case in which $r_1 = r_2 = \cdots = r_\nu \equiv r$, so that $N = r^\nu$. In such a case the DFTs are of size r, so that the computation of the N-point DFT has a regular pattern. The number r is called the radix of the FFT algorithm.

In this section we describe radix-2 algorithms, which are by far the most widely used FFT algorithms. Radix-4 algorithms are described in the following section.

Let us consider the computation of the $N = 2^\nu$ point DFT by the divide-and-conquer approach specified by (8.1.16) through (8.1.18). We select $M = N/2$ and $L = 2$. This selection results in a split of the N-point data sequence into two $N/2$-point data sequences $f_1(n)$ and $f_2(n)$, corresponding to the even-numbered and odd-numbered samples of $x(n)$, respectively, that is,

$$\begin{aligned} f_1(n) &= x(2n) \\ f_2(n) &= x(2n+1), \qquad n = 0, 1, \ldots, \frac{N}{2} - 1 \end{aligned} \tag{8.1.23}$$

Thus $f_1(n)$ and $f_2(n)$ are obtained by decimating $x(n)$ by a factor of 2, and hence the resulting FFT algorithm is called a decimation-in-time algorithm.

Now the N-point DFT can be expressed in terms of the DFTs of the decimated sequences as follows:

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) W_N^{kn}, \qquad k = 0, 1, \ldots, N-1 \\ &= \sum_{n \text{ even}} x(n) W_N^{kn} + \sum_{n \text{ odd}} x(n) W_N^{kn} \\ &= \sum_{m=0}^{(N/2)-1} x(2m) W_N^{2mk} + \sum_{m=0}^{(N/2)-1} x(2m+1) W_N^{k(2m+1)} \end{aligned} \tag{8.1.24}$$

But $W_N^2 = W_{N/2}$. With this substitution, (8.1.24) can be expressed as

$$\begin{aligned} X(k) &= \sum_{m=0}^{(N/2)-1} f_1(m) W_{N/2}^{km} + W_N^k \sum_{m=0}^{(N/2)-1} f_2(m) W_{N/2}^{km} \\ &= F_1(k) + W_N^k F_2(k), \qquad k = 0, 1, \ldots, N-1 \end{aligned} \tag{8.1.25}$$

where $F_1(k)$ and $F_2(k)$ are the $N/2$-point DFTs of the sequences $f_1(m)$ and $f_2(m)$, respectively.

Since $F_1(k)$ and $F_2(k)$ are periodic, with period $N/2$, we have $F_1(k + N/2) = F_1(k)$ and $F_2(k + N/2) = F_2(k)$. In addition, the factor $W_N^{k+N/2} = -W_N^k$. Hence (8.1.25) can be expressed as

$$X(k) = F_1(k) + W_N^k F_2(k), \qquad k = 0, 1, \ldots, \frac{N}{2} - 1 \tag{8.1.26}$$

$$X\left(k + \frac{N}{2}\right) = F_1(k) - W_N^k F_2(k), \qquad k = 0, 1, \ldots, \frac{N}{2} - 1 \tag{8.1.27}$$

We observe that the direct computation of $F_1(k)$ requires $(N/2)^2$ complex multiplications. The same applies to the computation of $F_2(k)$. Furthermore, there are $N/2$ additional complex multiplications required to compute $W_N^k F_2(k)$. Hence the computation of $X(k)$ requires $2(N/2)^2 + N/2 = N^2/2 + N/2$ complex multiplications. This first step results in a reduction of the number of multiplications from N^2 to $N^2/2 + N/2$, which is about a factor of 2 for N large.

To be consistent with our previous notation, we may define

$$G_1(k) = F_1(k) \quad , \quad k = 0, 1, \ldots, \frac{N}{2} - 1$$

$$G_2(k) = W_N^k F_2(k), \quad k = 0, 1, \ldots, \frac{N}{2} - 1$$

Then the DFT $X(k)$ may be expressed as

$$\begin{aligned} X(k) &= G_1(k) + G_2(k), \quad k = 0, 1, \ldots, \frac{N}{2} - 1 \\ X(k + \frac{N}{2}) &= G_1(k) - G_2(k), \quad k = 0, 1, \ldots, \frac{N}{2} - 1 \end{aligned} \tag{8.1.28}$$

This computation is illustrated in Fig. 8.1.4.

Having performed the decimation-in-time once, we can repeat the process for each of the sequences $f_1(n)$ and $f_2(n)$. Thus $f_1(n)$ would result in the two $N/4$-point sequences

$$\begin{aligned} v_{11}(n) &= f_1(2n), \qquad n = 0, 1, \ldots, \frac{N}{4} - 1 \\ v_{12}(n) &= f_1(2n + 1), \quad n = 0, 1, \ldots, \frac{N}{4} - 1 \end{aligned} \tag{8.1.29}$$

and $f_2(n)$ would yield

$$\begin{aligned} v_{21}(n) &= f_2(2n), \qquad n = 0, 1, \ldots, \frac{N}{4} - 1 \\ v_{22}(n) &= f_2(2n + 1), \quad n = 0, 1, \ldots, \frac{N}{4} - 1 \end{aligned} \tag{8.1.30}$$

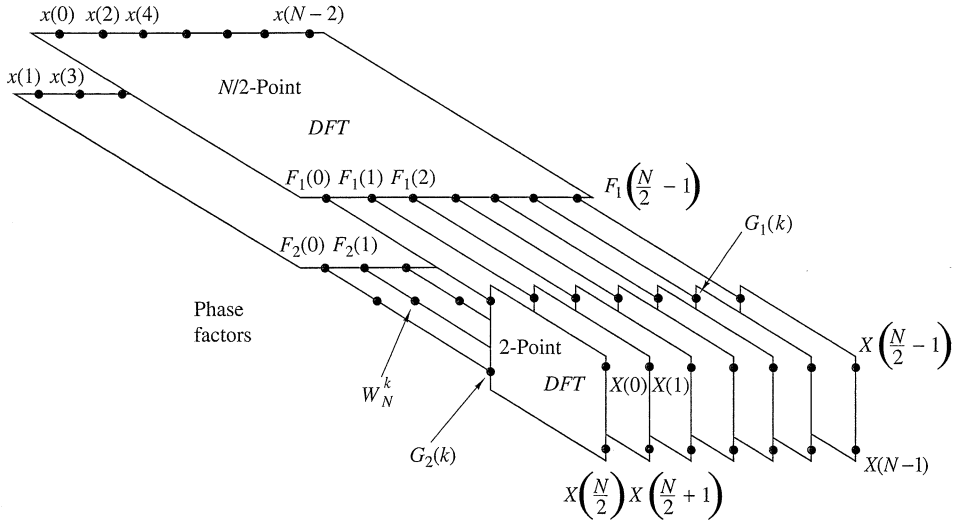


Figure 8.1.4 First step in the decimation-in-time algorithm.

By computing $N/4$-point DFTs, we would obtain the $N/2$-point DFTs $F_1(k)$ and $F_2(k)$ from the relations

$$F_1(k) = V_{11}(k) + W_{N/2}^k V_{12}(k), \quad k = 0, 1, \ldots, \frac{N}{4} - 1$$

$$F_1\left(k + \frac{N}{4}\right) = V_{11}(k) - W_{N/2}^k V_{12}(k), \quad k = 0, 1, \ldots, \frac{N}{4} - 1 \tag{8.1.31}$$

$$F_2(k) = V_{21}(k) + W_{N/2}^k V_{22}(k), \quad k = 0, 1, \ldots, \frac{N}{4} - 1$$

$$F_2\left(k + \frac{N}{4}\right) = V_{21}(k) - W_{N/2}^k V_{22}(k), \quad k = 0, \ldots, \frac{N}{4} - 1 \tag{8.1.32}$$

where the $\{V_{ij}(k)\}$ are the $N/4$-point DFTs of the sequences $\{v_{ij}(n)\}$.

We observe that the computation of $\{V_{ij}(k)\}$ requires $4(N/4)^2$ multiplications and hence the computation of $F_1(k)$ and $F_2(k)$ can be accomplished with $N^2/4 + N/2$ complex multiplications. An additional $N/2$ complex multiplications are required to compute $X(k)$ from $F_1(k)$ and $F_2(k)$. Consequently, the total number of multiplications is reduced approximately by a factor of 2 again to $N^2/4 + N$.

The decimation of the data sequence can be repeated again and again until the resulting sequences are reduced to one-point sequences. For $N = 2^\nu$, this decimation can be performed $\nu = \log_2 N$ times. Thus the total number of complex multiplications is reduced to $(N/2)\log_2 N$. The number of complex additions is $N \log_2 N$. Table 8.1 presents a comparison of the number of complex multiplications in the FFT and in the direct computation of the DFT.

For illustrative purposes, Fig. 8.1.5 depicts the computation of an $N = 8$-point DFT. We observe that the computation is performed in three stages, beginning with the computations of four two-point DFTs, then two four-point DFTs, and finally, one

TABLE 8.1 Comparison of Computational Complexity for the Direct Computation of the DFT Versus the FFT Algorithm

Number of Points, N	Complex Multiplications in Direct Computation, N^2	Complex Multiplications in FFT Algorithm, $(N/2)\log_2 N$	Speed Improvement Factor
4	16	4	4.0
8	64	12	5.3
16	256	32	8.0
32	1,024	80	12.8
64	4,096	192	21.3
128	16,384	448	36.6
256	65,536	1,024	64.0
512	262,144	2,304	113.8
1,024	1,048,576	5,120	204.8

eight-point DFT. The combination of the smaller DFTs to form the larger DFT is illustrated in Fig. 8.1.6 for $N = 8$.

Observe that the basic computation performed at every stage, as illustrated in Fig. 8.1.6, is to take two complex numbers, say the pair (a, b), multiply b by W_N^r, and then add and subtract the product from a to form two new complex numbers (A, B). This basic computation, which is shown in Fig. 8.1.7, is called a *butterfly* because the flow graph resembles a butterfly.

In general, each butterfly involves one complex multiplication and two complex additions. For $N = 2^\nu$, there are $N/2$ butterflies per stage of the computation process and $\log_2 N$ stages. Therefore, as previously indicated, the total number of complex multiplications is $(N/2)\log_2 N$ and complex additions is $N \log_2 N$.

Once a butterfly operation is performed on a pair of complex numbers (a, b) to produce (A, B), there is no need to save the input pair (a, b). Hence we can store the result (A, B) in the same locations as (a, b). Consequently, we require a fixed amount of storage, namely, $2N$ storage registers, in order to store the results

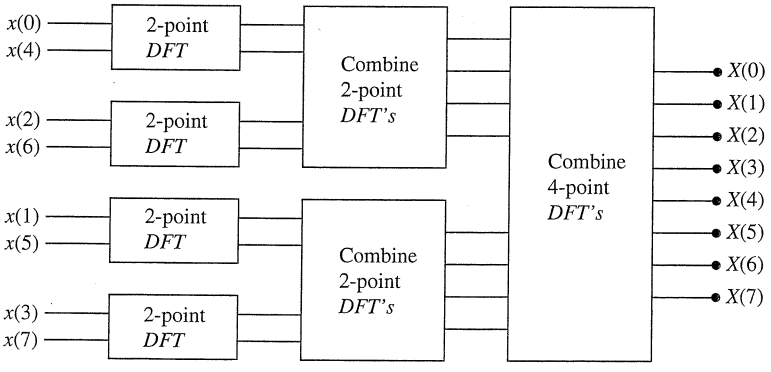


Figure 8.1.5 Three stages in the computation of an $N = 8$-point DFT.

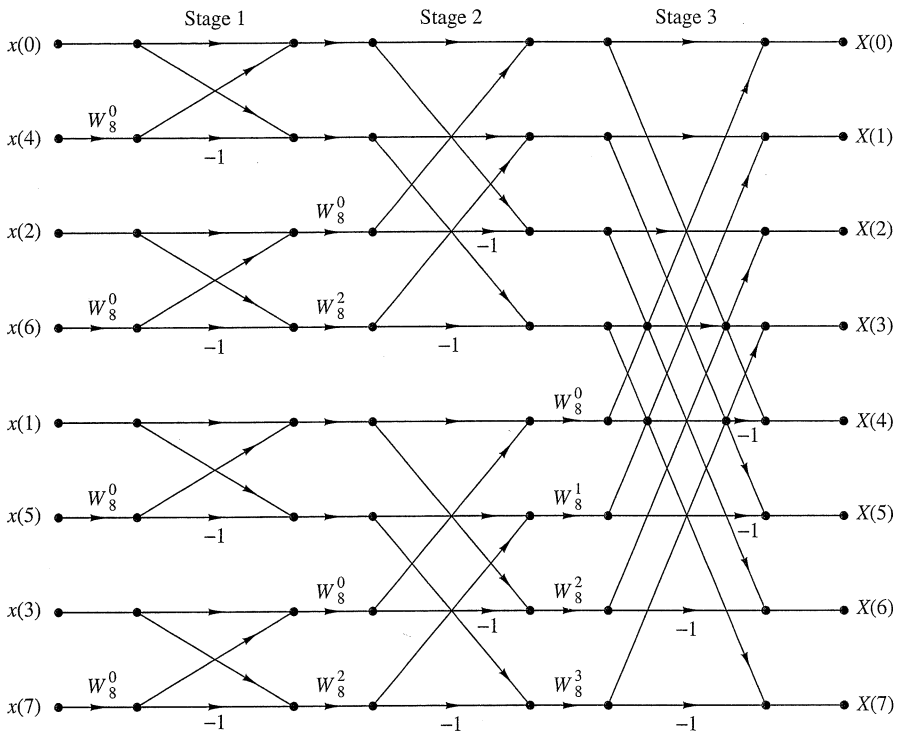


Figure 8.1.6 Eight-point decimation-in-time FFT algorithm.

(N complex numbers) of the computations at each stage. Since the same $2N$ storage locations are used throughout the computation of the N-point DFT, we say that *the computations are done in place.*

A second important observation is concerned with the order of the input data sequence after it is decimated $(\nu - 1)$ times. For example, if we consider the case where $N = 8$, we know that the first decimation yields the sequence $x(0), x(2), x(4), x(6), x(1), x(3), x(5), x(7)$, and the second decimation results in the sequence $x(0), x(4), x(2), x(6), x(1), x(5), x(3), x(7)$. This *shuffling* of the input data sequence has a well-defined order as can be ascertained from observing Fig. 8.1.8, which illustrates the decimation of the eight-point sequence. By expressing the index n, in the sequence $x(n)$, in binary form, we note that the order of the decimated data sequence is easily obtained by reading the binary representation of the index n in reverse order. Thus the data point $x(3) \equiv x(011)$ is placed in position $m = 110$ or $m = 6$ in the decimated array. Thus we say that the data $x(n)$ after decimation is stored in bit-reversed order.

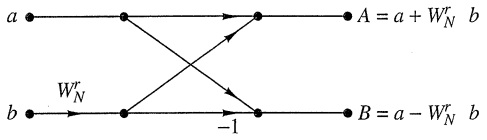
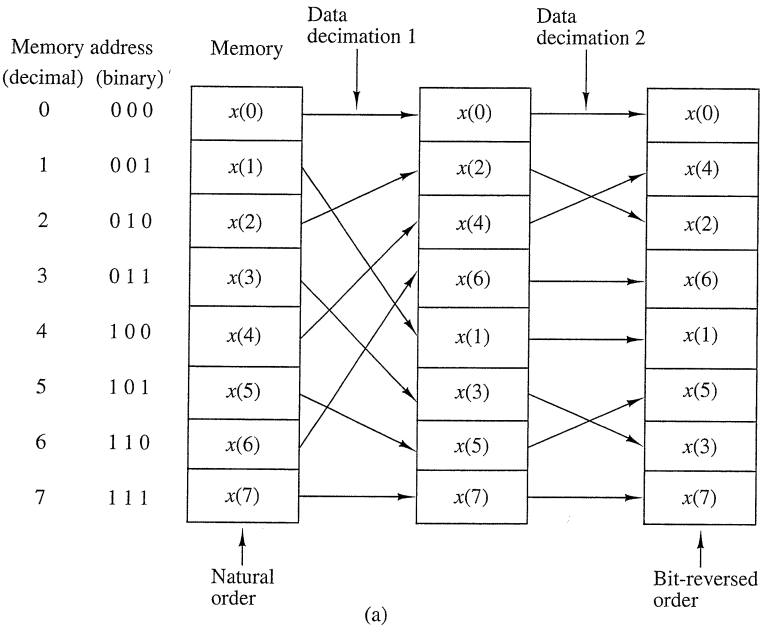


Figure 8.1.7
Basic butterfly computation in the decimation-in-time FFT algorithm.



(a)

$(n_2n_1n_0)$	$\rightarrow$	$(n_1n_0n_2)$	$\rightarrow$	$(n_0n_1n_2)$
(0 0 0)	$\rightarrow$	(0 0 0)	$\rightarrow$	(0 0 0)
(0 0 1)	$\rightarrow$	(0 1 0)	$\rightarrow$	(1 0 0)
(0 1 0)	$\rightarrow$	(1 0 0)	$\rightarrow$	(0 1 0)
(0 1 1)	$\rightarrow$	(1 1 0)	$\rightarrow$	(1 1 0)
(1 0 0)	$\rightarrow$	(0 0 1)	$\rightarrow$	(0 0 1)
(1 0 1)	$\rightarrow$	(0 1 1)	$\rightarrow$	(1 0 1)
(1 1 0)	$\rightarrow$	(1 0 1)	$\rightarrow$	(0 1 1)
(1 1 1)	$\rightarrow$	(1 1 1)	$\rightarrow$	(1 1 1)

(b)

Figure 8.1.8 Shuffling of the data and bit reversal.

With the input data sequence stored in bit-reversed order and the butterfly computations performed in place, the resulting DFT sequence $X(k)$ is obtained in natural order (i.e., $k = 0, 1, \ldots, N-1$). On the other hand, we should indicate that it is possible to arrange the FFT algorithm such that the input is left in natural order and the resulting output DFT will occur in bit-reversed order. Furthermore, we can impose the restriction that both the input data $x(n)$ and the output DFT $X(k)$ be in natural order, and derive an FFT algorithm in which the computations are not done in place. Hence such an algorithm requires additional storage.

Another important radix-2 FFT algorithm, called the decimation-in-frequency algorithm, is obtained by using the divide-and-conquer approach described in Section 8.1.2 with the choice of $M = 2$ and $L = N/2$. This choice of parameters implies a column-wise storage of the input data sequence. To derive the algorithm, we begin by splitting the DFT formula into two summations, of which one involves the sum over the first $N/2$ data points and the second the sum over the last $N/2$ data points.

Thus we obtain

$$\begin{aligned} X(k) &= \sum_{n=0}^{(N/2)-1} x(n)W_N^{kn} + \sum_{n=N/2}^{N-1} x(n)W_N^{kn} \\ &= \sum_{n=0}^{(N/2)-1} x(n)W_N^{kn} + W_N^{Nk/2} \sum_{n=0}^{(N/2)-1} x\left(n+\frac{N}{2}\right) W_N^{kn} \end{aligned} \tag{8.1.33}$$

Since $W_N^{kN/2} = (-1)^k$, the expression (8.1.33) can be rewritten as

$$X(k) = \sum_{n=0}^{(N/2)-1} \left[x(n) + (-1)^k x\left(n+\frac{N}{2}\right)\right] W_N^{kn} \tag{8.1.34}$$

Now, let us split (decimate) $X(k)$ into the even- and odd-numbered samples. Thus we obtain

$$X(2k) = \sum_{n=0}^{(N/2)-1} \left[x(n) + x\left(n+\frac{N}{2}\right)\right] W_{N/2}^{kn}, \qquad k = 0, 1, \ldots, \frac{N}{2} - 1 \tag{8.1.35}$$

and

$$X(2k+1) = \sum_{n=0}^{(N/2)-1} \left\{\left[x(n) - x\left(n+\frac{N}{2}\right)\right] W_N^n\right\} W_{N/2}^{kn}, \qquad k = 0, 1, \ldots, \frac{N}{2} - 1 \tag{8.1.36}$$

where we have used the fact that $W_N^2 = W_{N/2}$.

If we define the $N/2$-point sequences $g_1(n)$ and $g_2(n)$ as

$$\begin{aligned} g_1(n) &= x(n) + x\left(n+\frac{N}{2}\right) \\ g_2(n) &= \left[x(n) - x\left(n+\frac{N}{2}\right)\right] W_N^n, \qquad n = 0, 1, 2, \ldots, \frac{N}{2} - 1 \end{aligned} \tag{8.1.37}$$

then

$$\begin{aligned} X(2k) &= \sum_{n=0}^{(N/2)-1} g_1(n) W_{N/2}^{kn} \\ X(2k+1) &= \sum_{n=0}^{(N/2)-1} g_2(n) W_{N/2}^{kn} \end{aligned} \tag{8.1.38}$$

The computation of the sequences $g_1(n)$ and $g_2(n)$ according to (8.1.37) and the subsequent use of these sequences to compute the $N/2$-point DFTs are depicted in Fig. 8.1.9. We observe that the basic computation in this figure involves the butterfly operation illustrated in Fig. 8.1.10.

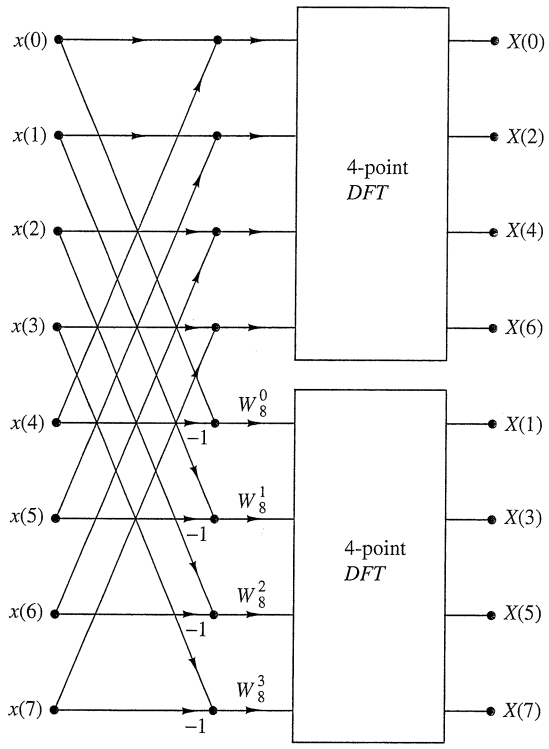


Figure 8.1.9
First stage of the decimation-in-frequency FFT algorithm.

This computational procedure can be repeated through decimation of the $N/2$-point DFTs, $X(2k)$ and $X(2k+1)$. The entire process involves $\nu = \log_2 N$ stages of decimation, where each stage involves $N/2$ butterflies of the type shown in Fig. 8.1.10. Consequently, the computation of the N-point DFT via the decimation-in-frequency FFT algorithm requires $(N/2)\log_2 N$ complex multiplications and $N \log_2 N$ complex additions, just as in the decimation-in-time algorithm. For illustrative purposes, the eight-point decimation-in-frequency algorithm is given in Fig. 8.1.11.

We observe from Fig. 8.1.11 that the input data $x(n)$ occurs in natural order, but the output DFT occurs in bit-reversed order. We also note that the computations are performed in place. However, it is possible to reconfigure the decimation-in-frequency algorithm so that the input sequence occurs in bit-reversed order while the output DFT occurs in normal order. Furthermore, if we abandon the requirement that the computations be done in place, it is also possible to have both the input data and the output DFT in normal order.

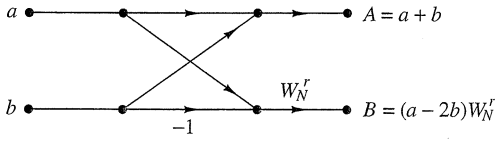


Figure 8.1.10
Basic butterfly computation in the decimation-in-frequency FFT algorithm.

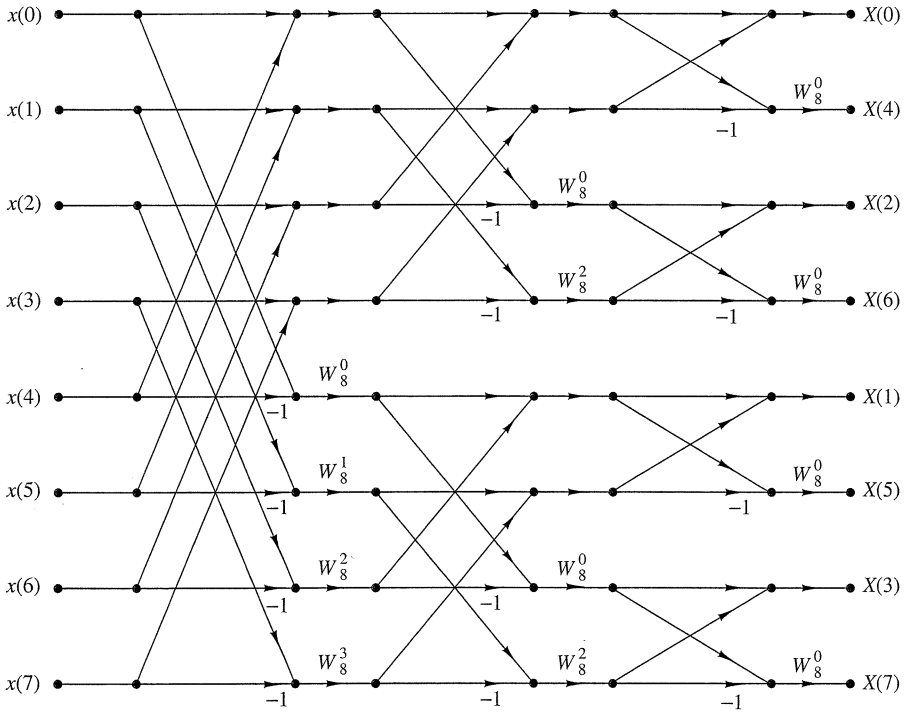


Figure 8.1.11 $N = 8$-point decimation-in-frequency FFT algorithm.

8.1.4 Radix-4 FFT Algorithms

When the number of data points N in the DFT is a power of 4 (i.e., $N = 4^\nu$), we can, of course, always use a radix-2 algorithm for the computation. However, for this case, it is more efficient computationally to employ a radix-4 FFT algorithm.

Let us begin by describing a radix-4 decimation-in-time FFT algorithm, which is obtained by selecting $L = 4$ and $M = N/4$ in the divide-and-conquer approach described in Section 8.1.2. For this choice of L and M, we have l, $p = 0, 1, 2, 3$; m, $q = 0, 1, \ldots, N/4 - 1$; $n = 4m + l$; and $k = (N/4)p + q$. Thus we split or decimate the N-point input sequence into four subsequences, $x(4n)$, $x(4n + 1)$, $x(4n + 2)$, $x(4n + 3)$, $n = 0, 1, \ldots, N/4 - 1$.

By applying (8.1.15) we obtain

$$X(p, q) = \sum_{l=0}^{3} \left[W_N^{lq} F(l, q) \right] W_4^{lp}, \qquad p = 0, 1, 2, 3 \tag{8.1.39}$$

where $F(l, q)$ is given by (8.1.16), that is,

$$F(l, q) = \sum_{m=0}^{(N/4)-1} x(l, m) W_{N/4}^{mq}, \qquad \begin{matrix} l = 0, 1, 2, 3, \\ q = 0, 1, 2, \ldots, \dfrac{N}{4} - 1 \end{matrix} \tag{8.1.40}$$

and

$$x(l, m) = x(4m + l) \tag{8.1.41}$$

$$X(p, q) = X\left(\frac{N}{4}p + q\right) \tag{8.1.42}$$

Thus, the four $N/4$-point DFTs obtained from (8.1.40) are combined according to (8.1.39) to yield the N-point DFT. The expression in (8.1.39) for combining the $N/4$-point DFTs defines a radix-4 decimation-in-time butterfly, which can be expressed in matrix form as

$$\begin{bmatrix} X(0,q) \\ X(1,q) \\ X(2,q) \\ X(3,q) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} W_N^0 F(0,q) \\ W_N^q F(1,q) \\ W_N^{2q} F(2,q) \\ W_N^{3q} F(3,q) \end{bmatrix} \tag{8.1.43}$$

The radix-4 butterfly is depicted in Fig. 8.1.12(a) and in a more compact form in Fig. 8.1.12(b). Note that since $W_N^0 = 1$, each butterfly involves three complex multiplications, and 12 complex additions.

This decimation-in-time procedure can be repeated recursively ν times. Hence the resulting FFT algorithm consists of ν stages, where each stage contains $N/4$ butterflies. Consequently, the computational burden for the algorithm is $3\nu N/4 = (3N/8)\log_2 N$ complex multiplications and $(3N/2)\log_2 N$ complex additions. We note that the number of multiplications is reduced by 25%, but the number of additions has increased by 50% from $N\log_2 N$ to $(3N/2)\log_2 N$.

It is interesting to note, however, that by performing the additions in two steps, it is possible to reduce the number of additions per butterfly from 12 to 8. This can

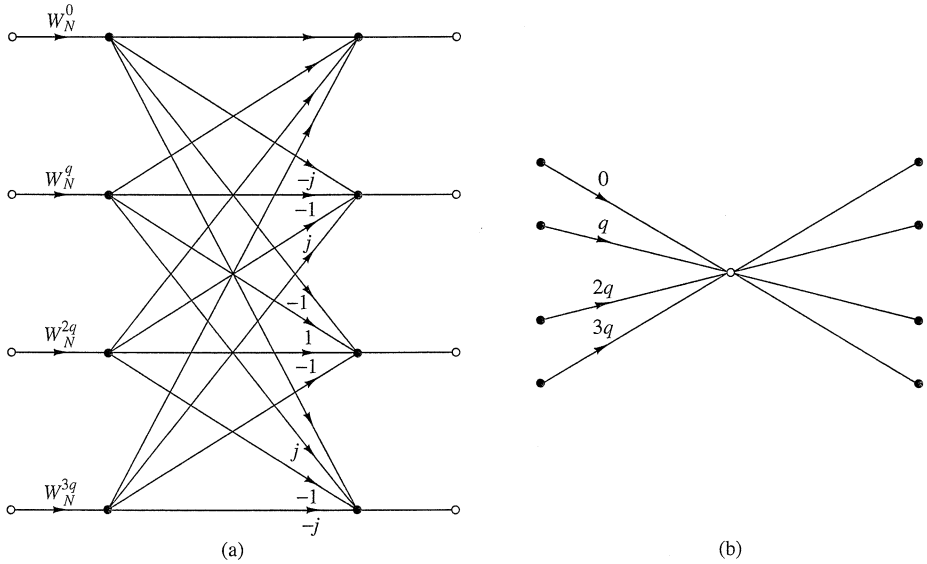


Figure 8.1.12 Basic butterfly computation in a radix-4 FFT algorithm.