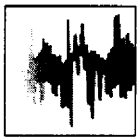


6

Efficient Computation of the DFT: Fast Fourier Transform Algorithms

As we have observed in the preceding chapter, the Discrete Fourier Transform (DFT) plays an important role in many applications of digital signal processing, including linear filtering, correlation analysis, and spectrum analysis. A major reason for its importance is the existence of efficient algorithms for computing the DFT.

The main topic of this chapter is the description of computationally efficient algorithms for evaluating the DFT. Two different approaches are described. One is a divide-and-conquer approach in which a DFT of size N, where N is a composite number, is reduced to the computation of smaller DFTs from which the larger DFT is computed. In particular, we present important computational algorithms, called fast Fourier transform (FFT) algorithms, for computing the DFT when the size N is a power of 2 and when it is a power of 4.

The second approach is based on the formulation of the DFT as a linear filtering operation on the data. This approach leads to two algorithms, the Goertzel algorithm and the chirp-z transform algorithm for computing the DFT via linear filtering of the data sequence.

6.1 EFFICIENT COMPUTATION OF THE DFT: FFT ALGORITHMS

In this section we present several methods for computing the DFT efficiently. In view of the importance of the DFT in various digital signal processing applications, such as linear filtering, correlation analysis, and spectrum analysis, its efficient computation is a topic that has received considerable attention by many mathematicians, engineers, and applied scientists.

Basically, the computational problem for the DFT is to compute the sequence $\{X(k)\}$ of N complex-valued numbers given another sequence of data $\{x(n)\}$ of

length N, according to the formula

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \qquad 0 \le k \le N-1 \tag{6.1.1}$$

where

$$W_N = e^{-j2\pi/N} \tag{6.1.2}$$

In general, the data sequence $x(n)$ is also assumed to be complex valued.

Similarly, the IDFT becomes

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-nk} \qquad 0 \le n \le N-1 \tag{6.1.3}$$

Since the DFT and IDFT involve basically the same type of computations, our discussion of efficient computational algorithms for the DFT applies as well to the efficient computation of the IDFT.

We observe that for each value of k, direct computation of $X(k)$ involves N complex multiplications ($4N$ real multiplications) and $N-1$ complex additions ($4N-2$ real additions). Consequently, to compute all N values of the DFT requires N^2 complex multiplications and $N^2 - N$ complex additions.

Direct computation of the DFT is basically inefficient primarily because it does not exploit the symmetry and periodicity properties of the phase factor W_N. In particular, these two properties are:

$$\text{Symmetry property:} \quad W_N^{k+N/2} = -W_N^k \tag{6.1.4}$$

$$\text{Periodicity property:} \quad W_N^{k+N} = W_N^k \tag{6.1.5}$$

The computationally efficient algorithms described in this section, known collectively as fast Fourier transform (FFT) algorithms, exploit these two basic properties of the phase factor.

6.1.1 Direct Computation of the DFT

For a complex-valued sequence $x(n)$ of N points, the DFT may be expressed as

$$X_R(k) = \sum_{n=0}^{N-1} \left[x_R(n) \cos \frac{2\pi kn}{N} + x_I(n) \sin \frac{2\pi kn}{N} \right] \tag{6.1.6}$$

$$X_I(k) = -\sum_{n=0}^{N-1} \left[x_R(n) \sin \frac{2\pi kn}{N} - x_I(n) \cos \frac{2\pi kn}{N} \right] \tag{6.1.7}$$

The direct computation of (6.1.6) and (6.1.7) requires:

1. $2N^2$ evaluations of trigonometric functions.
2. $4N^2$ real multiplications.

3. $4N(N-1)$ real additions.
4. A number of indexing and addressing operations.

These operations are typical of DFT computational algorithms. The operations in items 2 and 3 result in the DFT values $X_R(k)$ and $X_I(k)$. The indexing and addressing operations are necessary to fetch the data $x(n)$, $0 \le n \le N-1$, and the phase factors and to store the results. The variety of DFT algorithms optimize each of these computational processes in a different way.

6.1.2 Divide-and-Conquer Approach to Computation of the DFT

The development of computationally efficient algorithms for the DFT is made possible if we adopt a divide-and-conquer approach. This approach is based on the decomposition of an N-point DFT into successively smaller DFTs. This basic approach leads to a family of computationally efficient algorithms known collectively as FFT algorithms.

To illustrate the basic notions, let us consider the computation of an N-point DFT, where N can be factored as a product of two integers, that is,

$$N = LM \tag{6.1.8}$$

The assumption that N is not a prime number is not restrictive, since we can pad any sequence with zeros to ensure a factorization of the form (6.1.8).

Now the sequence $x(n)$, $0 \le n \le N-1$, can be stored in either a one-dimensional array indexed by n or as a two-dimensional array indexed by l and m, where $0 \le l \le L-1$ and $0 \le m \le M-1$ as illustrated in Fig. 6.1. Note that l is the row index and m is the column index. Thus, the sequence $x(n)$ can be stored in a rectangular array in a variety of ways, each of which depends on the mapping of index n to the indexes (l, m).

For example, suppose that we select the mapping

$$n = Ml + m \tag{6.1.9}$$

This leads to an arrangement in which the first row consists of the first M elements of $x(n)$, the second row consists of the next M elements of $x(n)$, and so on, as illustrated in Fig. 6.2(a). On the other hand, the mapping

$$n = l + mL \tag{6.1.10}$$

stores the first L elements of $x(n)$ in the first column, the next L elements in the second column, and so on, as illustrated in Fig. 6.2(b).

A similar arrangement can be used to store the computed DFT values. In particular, the mapping is from the index k to a pair of indices (p, q), where $0 \le p \le L-1$ and $0 \le q \le M-1$. If we select the mapping

$$k = Mp + q \tag{6.1.11}$$

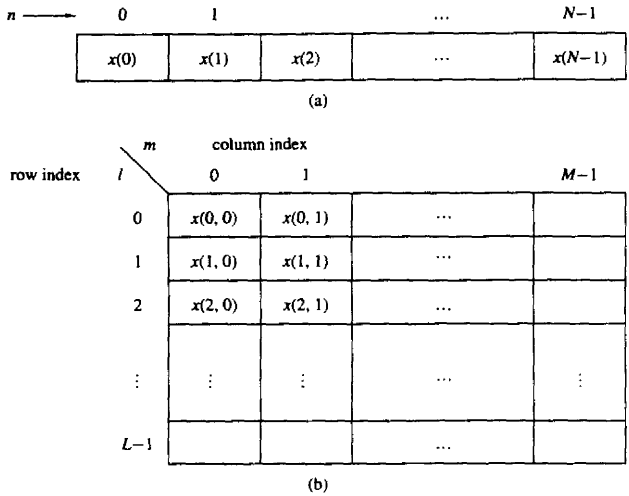


Figure 6.1 Two dimensional data array for storing the sequence $x(n)$, $0 \leq n \leq N-1$.

the DFT is stored on a row-wise basis, where the first row contains the first M elements of the DFT $X(k)$, the second row contains the next set of M elements, and so on. On the other hand, the mapping

$$k = qL + p \tag{6.1.12}$$

results in a column-wise storage of $X(k)$, where the first L elements are stored in the first column, the second set of L elements are stored in the second column, and so on.

Now suppose that $x(n)$ is mapped into the rectangular array $x(l, m)$ and $X(k)$ is mapped into a corresponding rectangular array $X(p, q)$. Then the DFT can be expressed as a double sum over the elements of the rectangular array multiplied by the corresponding phase factors. To be specific, let us adopt a column-wise mapping for $x(n)$ given by (6.1.10) and the row-wise mapping for the DFT given by (6.1.11). Then

$$X(p, q) = \sum_{m=0}^{M-1} \sum_{l=0}^{L-1} x(l, m) W_N^{(Mp+q)(mL+l)} \tag{6.1.13}$$

But

$$W_N^{(Mp+q)(mL+l)} = W_N^{MLmp} W_N^{mLq} W_N^{Mpl} W_N^{lq} \tag{6.1.14}$$

However, $W_N^{NmP} = 1$, $W_N^{mqL} = W_{N/L}^{mq} = W_M^{mq}$, and $W_N^{Mpl} = W_{N/M}^{pl} = W_L^{pl}$.

Row-wise $n = Ml + m$

l \ m	0	1	2	...	$M-1$
0	$x(0)$	$x(1)$	$x(2)$	...	$x(M-1)$
1	$x(M)$	$x(M+1)$	$x(M+2)$	...	$x(2M-1)$
2	$x(2M)$	$x(2M+1)$	$x(2M+2)$	...	$x(3M-1)$
	⋮	⋮	⋮	...	⋮
$L-1$	$x((L-1)M)$	$x((L-1)M+1)$	$x((L-1)M+2)$	...	$x(LM-1)$

(a)

Column-wise $n = l + mL$

l \ m	0	1	2	...	$M-1$
0	$x(0)$	$x(L)$	$x(2L)$	...	$x((M-1)L)$
1	$x(1)$	$x(L+1)$	$x(2L+1)$	...	$x((M-1)L+1)$
2	$x(2)$	$x(L+2)$	$x(2L+2)$	...	$x((M-1)L+2)$
	⋮	⋮	⋮	...	⋮
$L-1$	$x(L-1)$	$x(2L-1)$	$x(3L-1)$	...	$x(LM-1)$

(b)

Figure 6.2 Two arrangements for the data arrays.

With these simplifications, (6.1.13) can be expressed as

$$X(p,q) = \sum_{l=0}^{L-1} \left\{ W_N^{lq} \left[\sum_{m=0}^{M-1} x(l,m) W_M^{mq} \right] \right\} W_L^{lp} \tag{6.1.15}$$

The expression in (6.1.15) involves the computation of DFTs of length M and length L. To elaborate, let us subdivide the computation into three steps:

1. First, we compute the M-point DFTs

$$F(l,q) \equiv \sum_{m=0}^{M-1} x(l,m) W_M^{mq}, \qquad 0 \le q \le M-1 \tag{6.1.16}$$

for each of the rows $l = 0, 1, \ldots, L-1$.

2. Second, we compute a new rectangular array $G(l, q)$ defined as

$$G(l, q) = W_N^{lq} F(l, q) \qquad \begin{matrix} 0 \le l \le L-1 \\ 0 \le q \le M-1 \end{matrix} \tag{6.1.17}$$

3. Finally, we compute the L-point DFTs

$$X(p, q) = \sum_{l=0}^{L-1} G(l, q) W_L^{lp} \tag{6.1.18}$$

for each column $q = 0, 1, \ldots, M-1$, of the array $G(l, q)$.

On the surface it may appear that the computational procedure outlined above is more complex than the direct computation of the DFT. However, let us evaluate the computational complexity of (6.1.15). The first step involves the computation of L DFTs, each of M points. Hence this step requires LM^2 complex multiplications and $LM(M-1)$ complex additions. The second step requires LM complex multiplications. Finally, the third step in the computation requires ML^2 complex multiplications and $ML(L-1)$ complex additions. Therefore, the computational complexity is

$$\begin{matrix} \text{Complex multiplications:} & N(M+L+1) \\ \text{Complex additions:} & N(M+L-2) \end{matrix} \tag{6.1.19}$$

where $N = ML$. Thus the number of multiplications has been reduced from N^2 to $N(M+L+1)$ and the number of additions has been reduced from $N(N-1)$ to $N(M+L-2)$.

For example, suppose that $N = 1000$ and we select $L = 2$ and $M = 500$. Then, instead of having to perform 10^6 complex multiplications via direct computation of the DFT, this approach leads to 503,000 complex multiplications. This represents a reduction by approximately a factor of 2. The number of additions is also reduced by about a factor of 2.

When N is a highly composite number, that is, N can be factored into a product of prime numbers of the form

$$N = r_1 r_2 \cdots r_v \tag{6.1.20}$$

then the decomposition above can be repeated $(v-1)$ more times. This procedure results in smaller DFTs, which, in turn, leads to a more efficient computational algorithm.

In effect, the first segmentation of the sequence $x(n)$ into a rectangular array of M columns with L elements in each column resulted in DFTs of sizes L and M. Further decomposition of the data in effect involves the segmentation of each row (or column) into smaller rectangular arrays which result in smaller DFTs. This procedure terminates when N is factored into its prime factors.

Example 6.1.1

To illustrate this computational procedure, let us consider the computation of an $N = 15$ point DFT. Since $N = 5 \times 3 = 15$, we select $L = 5$ and $M = 3$. In other

words, we store the 15-point sequence $x(n)$ column-wise as follows:

Row 1:	$x(0, 0) = x(0)$	$x(0, 1) = x(5)$	$x(0, 2) = x(10)$
Row 2:	$x(1, 0) = x(1)$	$x(1, 1) = x(6)$	$x(1, 2) = x(11)$
Row 3:	$x(2, 0) = x(2)$	$x(2, 1) = x(7)$	$x(2, 2) = x(12)$
Row 4:	$x(3, 0) = x(3)$	$x(3, 1) = x(8)$	$x(3, 2) = x(13)$
Row 5:	$x(4, 0) = x(4)$	$x(4, 1) = x(9)$	$x(4, 2) = x(14)$

Now, we compute the three-point DFTs for each of the five rows. This leads to the following 5×3 array:

$F(0, 0)$	$F(0, 1)$	$F(0, 2)$
$F(1, 0)$	$F(1, 1)$	$F(1, 2)$
$F(2, 0)$	$F(2, 1)$	$F(2, 2)$
$F(3, 0)$	$F(3, 1)$	$F(3, 2)$
$F(4, 0)$	$F(4, 1)$	$F(4, 2)$

The next step is to multiply each of the terms $F(l, q)$ by the phase factors $W_N^{lq} = W_{15}^{lq}$, $0 \le l \le 4$ and $0 \le q \le 2$. This computation results in the 5×3 array:

Column 1	Column 2	Column 3
$G(0, 0)$	$G(0, 1)$	$G(0, 2)$
$G(1, 0)$	$G(1, 1)$	$G(1, 2)$
$G(2, 0)$	$G(2, 1)$	$G(2, 2)$
$G(3, 0)$	$G(3, 1)$	$G(3, 2)$
$G(4, 0)$	$G(4, 1)$	$G(4, 2)$

The final step is to compute the five-point DFTs for each of the three columns. This computation yields the desired values of the DFT in the form

$X(0, 0) = X(0)$	$X(0, 1) = X(1)$	$X(0, 2) = X(2)$
$X(1, 0) = X(3)$	$X(1, 1) = X(4)$	$X(1, 2) = X(5)$
$X(2, 0) = X(6)$	$X(2, 1) = X(7)$	$X(2, 2) = X(8)$
$X(3, 0) = X(9)$	$X(3, 1) = X(10)$	$X(3, 2) = X(11)$
$X(4, 0) = X(12)$	$X(4, 1) = X(13)$	$X(4, 2) = X(14)$

Figure 6.3 illustrates the steps in the computation.

It is interesting to view the segmented data sequence and the resulting DFT in terms of one-dimensional arrays. When the input sequence $x(n)$ and the output DFT $X(k)$ in the two-dimensional arrays are read across from row 1 through row 5, we obtain the following sequences:

INPUT ARRAY

$x(0)\ x(5)\ x(10)\ x(1)\ x(6)\ x(11)\ x(2)\ x(7)\ x(12)\ x(3)\ x(8)\ x(13)\ x(4)\ x(9)\ x(14)$

OUTPUT ARRAY

$X(0)\ X(1)\ X(2)\ X(3)\ X(4)\ X(5)\ X(6)\ X(7)\ X(8)\ X(9)\ X(10)\ X(11)\ X(12)\ X(13)\ X(14)$

We observe that the input data sequence is shuffled from the normal order in the computation of the DFT. On the other hand, the output sequence occurs in normal order. In this case the rearrangement of the input data array is due to the

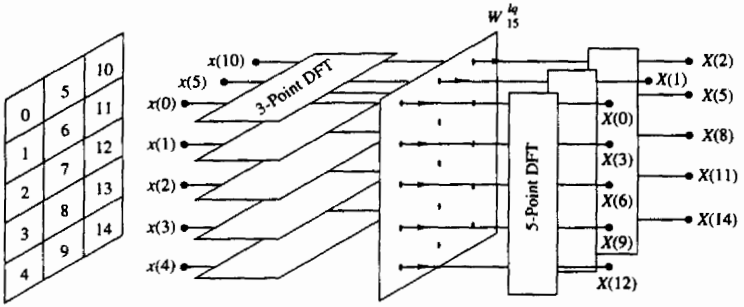


Figure 6.3 Computation of $N = 15$-point DFT by means of 3-point and 5-point DFTs.

segmentation of the one-dimensional array into a rectangular array and the order in which the DFTs are computed. This shuffling of either the input data sequence or the output DFT sequence is a characteristic of most FFT algorithms.

To summarize, the algorithm that we have introduced involves the following computations:

Algorithm 1

1. Store the signal column-wise.
2. Compute the M-point DFT of each row.
3. Multiply the resulting array by the phase factors W_N^{lq}.
4. Compute the L-point DFT of each column
5. Read the resulting array row-wise.

An additional algorithm with a similar computational structure can be obtained if the input signal is stored row-wise and the resulting transformation is. column-wise. In this case we select as

$$\begin{aligned} n &= Ml + m \\ k &= qL + p \end{aligned} \tag{6.1.21}$$

This choice of indices leads to the formula for the DFT in the form

$$\begin{aligned} X(p,q) &= \sum_{m=0}^{M-1} \sum_{l=0}^{L-1} x(l,m) W_N^{pm} W_L^{pl} W_M^{qm} \\ &= \sum_{m=0}^{M-1} W_M^{mq} \left[\sum_{l=0}^{L-1} x(l,m) W_L^{lp} \right] W_N^{mp} \end{aligned} \tag{6.1.22}$$

Thus we obtain a second algorithm.

Algorithm 2

1. Store the signal row-wise.
2. Compute the L-point DFT at each column.
3. Multiply the resulting array by the factors W_N^{pm}.
4. Compute the M-point DFT of each row.
5. Read the resulting array column-wise.

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.

6.1.3 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 as 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 (6.1.16) through (6.1.18). 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,

$$\begin{aligned} f_1(n) &= x(2n) \\ f_2(n) &= x(2n+1), \qquad n = 0, 1, \ldots, \frac{N}{2} - 1 \end{aligned} \tag{6.1.23}$$

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:

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

$$= \sum_{n \text{ even}} x(n) W_N^{kn} + \sum_{n \text{ odd}} x(n) W_N^{kn} \tag{6.1.24}$$

$$= \sum_{m=0}^{(N/2)-1} x(2m) W_N^{2mk} + \sum_{m=0}^{(N/2)-1} x(2m+1) W_N^{k(2m+1)}$$

But $W_N^2 = W_{N/2}$. With this substitution, (6.1.24) can be expressed as

$$\begin{aligned} X(k) &= \sum_{m=0}^{(N/2)-1} f_1(m) W_{N/2}^{km} + W_N^k \sum_{m=0}^{(N/2)-1} f_2(m) W_{N/2}^{km} \\ &= F_1(k) + W_N^k F_2(k) \qquad k = 0, 1, \ldots, N-1 \end{aligned} \tag{6.1.25}$$

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 (6.1.25) can be expressed as

$$X(k) = F_1(k) + W_N^k F_2(k) \qquad k = 0, 1, \ldots, \frac{N}{2} - 1 \tag{6.1.26}$$

$$X\left(k + \frac{N}{2}\right) = F_1(k) - W_N^k F_2(k) \qquad k = 0, 1, \ldots, \frac{N}{2} - 1 \tag{6.1.27}$$

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

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

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

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

$$\begin{aligned} X(k) &= G_1(k) + G_2(k) \qquad k = 0, 1, \ldots, \frac{N}{2} - 1 \\ X(k + \frac{N}{2}) &= G_1(k) - G_2(k) \qquad k = 0, 1, \ldots, \frac{N}{2} - 1 \end{aligned} \tag{6.1.28}$$

This computation is illustrated in Fig. 6.4.

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

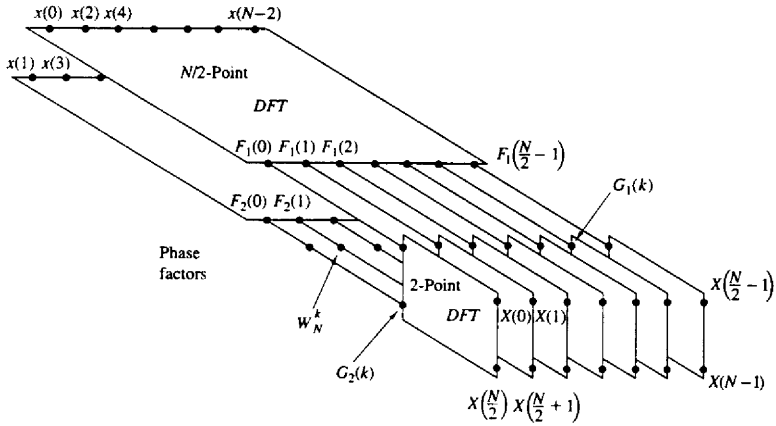


Figure 6.4 First step in the decimation-in-time algorithm.

$N/4$-point sequences

$$\begin{aligned} v_{11}(n) &= f_1(2n) \qquad n = 0, 1, \ldots, \frac{N}{4} - 1 \\ v_{12}(n) &= f_1(2n+1) \qquad n = 0, 1, \ldots, \frac{N}{4} - 1 \end{aligned} \tag{6.1.29}$$

and $f_2(n)$ would yield

$$\begin{aligned} v_{21}(n) &= f_2(2n) \qquad n = 0, 1, \ldots, \frac{N}{4} - 1 \\ v_{22}(n) &= f_2(2n+1) \qquad n = 0, 1, \ldots, \frac{N}{4} - 1 \end{aligned} \tag{6.1.30}$$

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

$$\begin{aligned} F_1(k) &= V_{11}(k) + W_{N/2}^k V_{12}(k) \qquad k = 0, 1, \ldots, \frac{N}{4} - 1 \\ F_1\left(k + \frac{N}{4}\right) &= V_{11}(k) - W_{N/2}^k V_{12}(k) \qquad k = 0, 1, \ldots, \frac{N}{4} - 1 \end{aligned} \tag{6.1.31}$$

$$\begin{aligned} F_2(k) &= V_{21}(k) + W_{N/2}^k V_{22}(k) \qquad k = 0, 1, \ldots, \frac{N}{4} - 1 \\ F_2\left(k + \frac{N}{4}\right) &= V_{21}(k) - W_{N/2}^k V_{22}(k) \qquad k = 0, \ldots, \frac{N}{4} - 1 \end{aligned} \tag{6.1.32}$$

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

TABLE 6.1 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

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 $N = 2^v$, this decimation can be performed $v = \log_2 N$ 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 6.1 presents a comparison of the number of complex multiplications in the FFT and in the direct computation of the DFT.

For illustrative purposes, Fig. 6.5 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

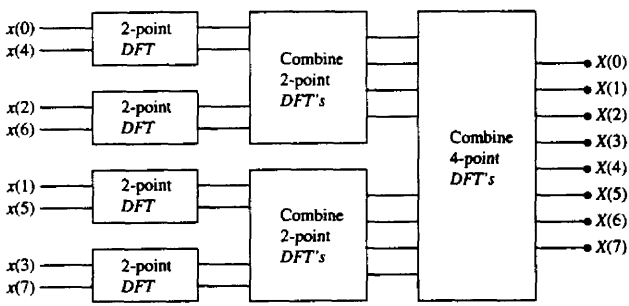


Figure 6.5 Three stages in the computation of an $N = 8$-point DFT.

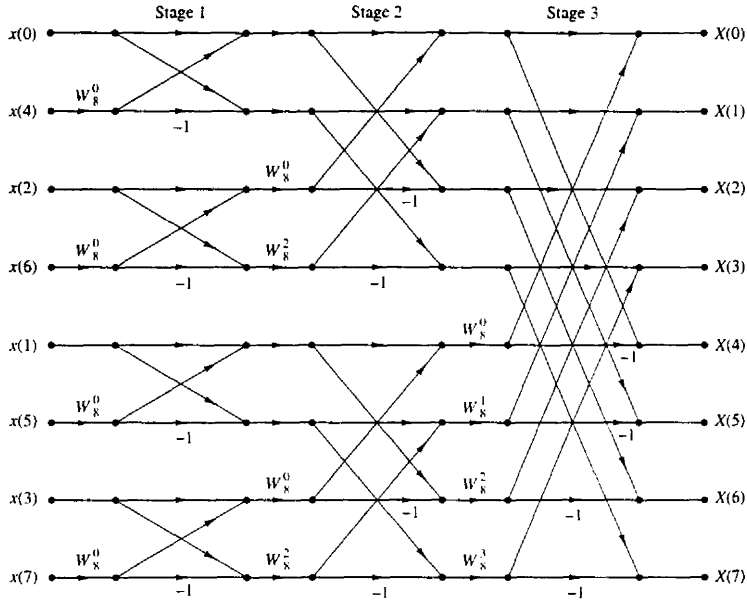


Figure 6.6 Eight-point decimation-in-time FFT algorithm.

finally, one eight-point DFT. The combination of the smaller DFTs to form the larger DFT is illustrated in Fig. 6.6 for $N = 8$.

Observe that the basic computation performed at every stage, as illustrated in Fig. 6.6, 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. 6.7, 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

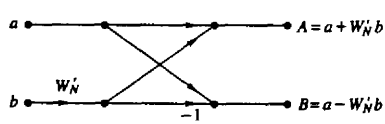


Figure 6.7 Basic butterfly computation in the decimation-in-time FFT algorithm.

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 (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. 6.8, 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.

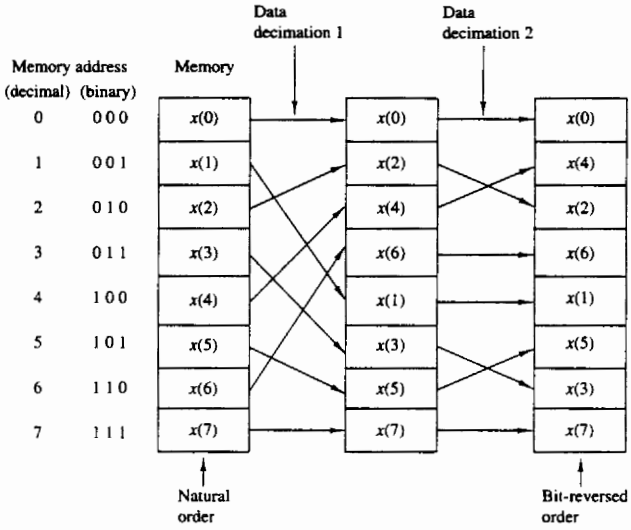
With the input data sequence stored in bit-reversed order and the butterfly computations performed in place, the resulting DFT sequence $X(k)$ is obtained in natural order (i.e., $k = 0, 1, \ldots, N - 1$). On the other hand, we should indicate that it is possible to arrange the FFT algorithm such that 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 6.1.2 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, one of which involves the sum over the first $N/2$ data points and the second sum involves the last $N/2$ data points. Thus we obtain

$$\begin{aligned} X(k) &= \sum_{n=0}^{(N/2)-1} x(n) W_N^{kn} + \sum_{n=N/2}^{N-1} x(n) W_N^{kn} \\ &= \sum_{n=0}^{(N/2)-1} x(n) W_N^{kn} + W_N^{Nk/2} \sum_{n=0}^{(N/2)-1} x\left(n + \frac{N}{2}\right) W_N^{kn} \end{aligned} \tag{6.1.33}$$

Since $W_N^{kN/2} = (-1)^k$, the expression (6.1.33) can be rewritten as

$$X(k) = \sum_{n=0}^{(N/2)-1} \left[x(n) + (-1)^k x\left(n + \frac{N}{2}\right) \right] W_N^{kn} \tag{6.1.34}$$



(a)

$(n_2n_1n_0)$	→	$(n_0n_2n_1)$	→	$(n_0n_1n_2)$
(0 0 0)	→	(0 0 0)	→	(0 0 0)
(0 0 1)	→	(1 0 0)	→	(1 0 0)
(0 1 0)	→	(0 0 1)	→	(0 1 0)
(0 1 1)	→	(1 0 1)	→	(1 1 0)
(1 0 0)	→	(0 1 0)	→	(0 0 1)
(1 0 1)	→	(1 1 0)	→	(1 0 1)
(1 1 0)	→	(0 1 1)	→	(0 1 1)
(1 1 1)	→	(1 1 1)	→	(1 1 1)

(b)

Figure 6.8 Shuffling of the data and bit reversal.

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

$$X(2k) = \sum_{n=0}^{(N/2)-1} \left[x(n) + x\left(n + \frac{N}{2}\right) \right] W_{N/2}^{kn} \qquad k = 0, 1, \ldots, \frac{N}{2} - 1 \tag{6.1.35}$$

and

$$X(2k+1) = \sum_{n=0}^{(N/2)-1} \left\{ \left[x(n) - x\left(n + \frac{N}{2}\right) \right] W_N^n \right\} W_{N/2}^{kn} \qquad k = 0, 1, \ldots, \frac{N}{2} - 1 \tag{6.1.36}$$

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

$$\begin{aligned} g_1(n) &= x(n) + x\left(n + \frac{N}{2}\right) \\ g_2(n) &= \left[x(n) - x\left(n + \frac{N}{2}\right)\right] W_N^n \qquad n = 0, 1, 2, \ldots, \frac{N}{2} - 1 \end{aligned} \tag{6.1.37}$$

then

$$\begin{aligned} X(2k) &= \sum_{n=0}^{(N/2)-1} g_1(n) W_{N/2}^{kn} \\ X(2k+1) &= \sum_{n=0}^{(N/2)-1} g_2(n) W_{N/2}^{kn} \end{aligned} \tag{6.1.38}$$

The computation of the sequences $g_1(n)$ and $g_2(n)$ according to (6.1.37) and the subsequent use of these sequences to compute the $N/2$-point DFTs are depicted in Fig. 6.9. We observe that the basic computation in this figure involves the butterfly operation illustrated in Fig. 6.10.

This computational procedure can be repeated through decimation of the $N/2$-point DFTs, $X(2k)$ and $X(2k+1)$. The entire process involves $\nu = \log_2 N$

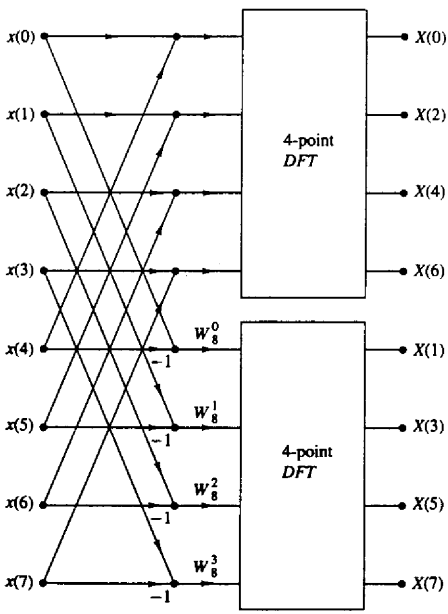


Figure 6.9 First stage of the decimation-in-frequency FFT algorithm.

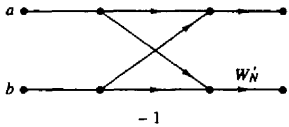


Figure 6.10 Basic butterfly computation in the decimation-in-frequency FFT algorithm.

stages of decimation, where each stage involves $N/2$ butterflies of the type shown in Fig. 6.10. 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. 6.11.

We observe from Fig. 6.11, 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.

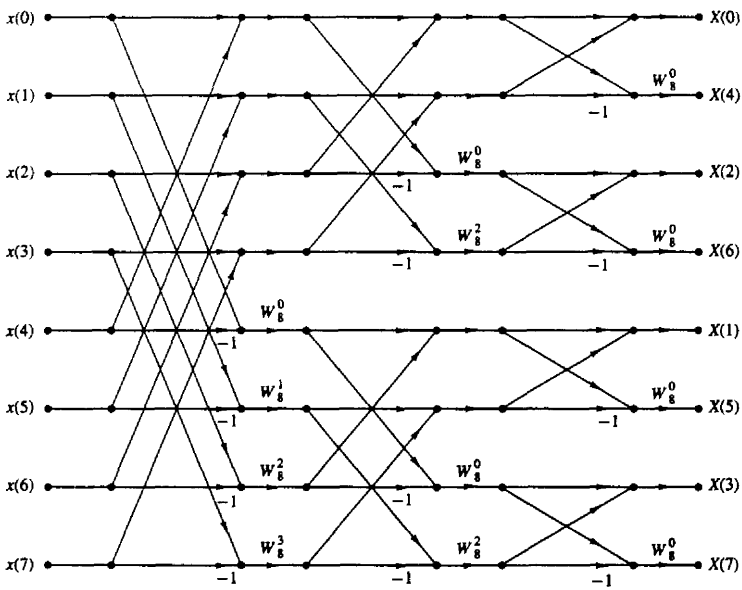


Figure 6.11 N = 8-point decimation-in-frequency FFT algorithmn.