6.1.4 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 6.1.2. 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 (6.1.15) we obtain

$$X(p, q) = \sum_{l=0}^{3} \left[W_N^{lq} F(l, q) \right] W_4^{lp} \qquad p = 0, 1, 2, 3 \tag{6.1.39}$$

where $F(l, q)$ is given by (6.1.16), that is,

$$F(l, q) = \sum_{m=0}^{(N/4)-1} x(l, m) W_{N/4}^{mq} \qquad \begin{matrix} l = 0, 1, 2, 3, \\ q = 0, 1, 2, \ldots, \dfrac{N}{4} - 1 \end{matrix} \tag{6.1.40}$$

and

$$x(l, m) = x(4m + l) \tag{6.1.41}$$

$$X(p, q) = X\left(\frac{N}{4}p + q\right) \tag{6.1.42}$$

Thus, the four $N/4$-point DFTs obtained from (6.1.40) are combined according to (6.1.39) to yield the N-point DFT. The expression in (6.1.39) for combining the $N/4$-point DFTs defines a radix-4 decimation-in-time butterfly, which can be expressed in matrix form as

$$\begin{bmatrix} X(0, q) \\ X(1, q) \\ X(2, q) \\ X(3, q) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} W_N^0 F(0, q) \\ W_N^q F(1, q) \\ W_N^{2q} F(2, q) \\ W_N^{3q} F(3, q) \end{bmatrix} \tag{6.1.43}$$

The radix-4 butterfly is depicted in Fig. 6.12(a) and in a more compact form in Fig. 6.12(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 $(3N/2) \log_2 N$.

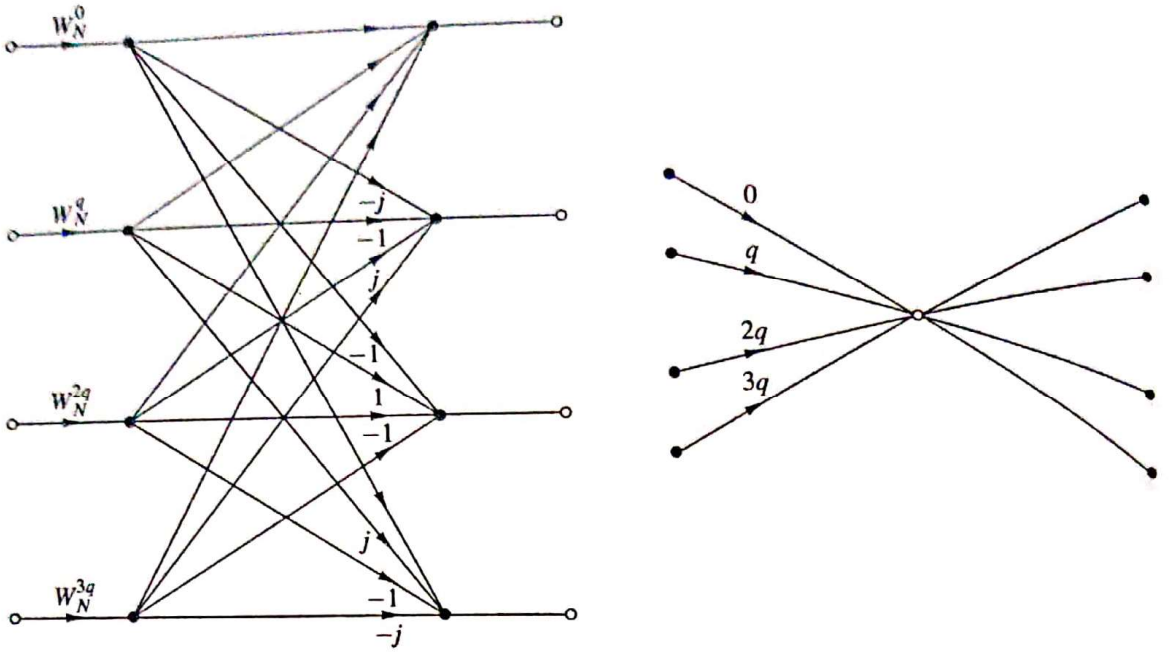

Figure 6.12 Basic butterfly computation in a radix-4 FFT algorithm.

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 be accomplished by expressing the matrix of the linear transformation in (6.1.43) as a product of two matrices as follows:

$$\begin{bmatrix} X(0,q) \\ X(1,q) \\ X(2,q) \\ X(3,q) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} W_N^0 F(0,q) \\ W_N^q F(1,q) \\ W_N^{2q} F(2,q) \\ W_N^{3q} F(3,q) \end{bmatrix} \tag{6.1.44}$$

Now each matrix multiplication involves four additions for a total of eight additions. Thus the total number of complex additions is reduced to $N \log_2 N$, which is identical to the radix-2 FFT algorithm. The computational savings results from the 25% reduction in the number of complex multiplications.

An illustration of a radix-4 decimation-in-time FFT algorithm is shown in Fig. 6.13 for $N = 16$. Note that in this algorithm, the input sequence is in normal order while the output DFT is shuffled. In the radix-4 FFT algorithm, where the decimation is by a factor of 4, the order of the decimated sequence can be determined by reversing the order of the number that represents the index n in a quaternary number system (i.e., the number system based on the digits 0, 1, 2, 3).

A radix-4 decimation-in-frequency FFT algorithm can be obtained by selecting $L = N/4$, $M = 4$; l, $p = 0, 1, \ldots, N/4 - 1$; m, $q = 0, 1, 2, 3$; $n = (N/4)m + l$; and $k = 4p + q$. With this choice of parameters, the general equation given by

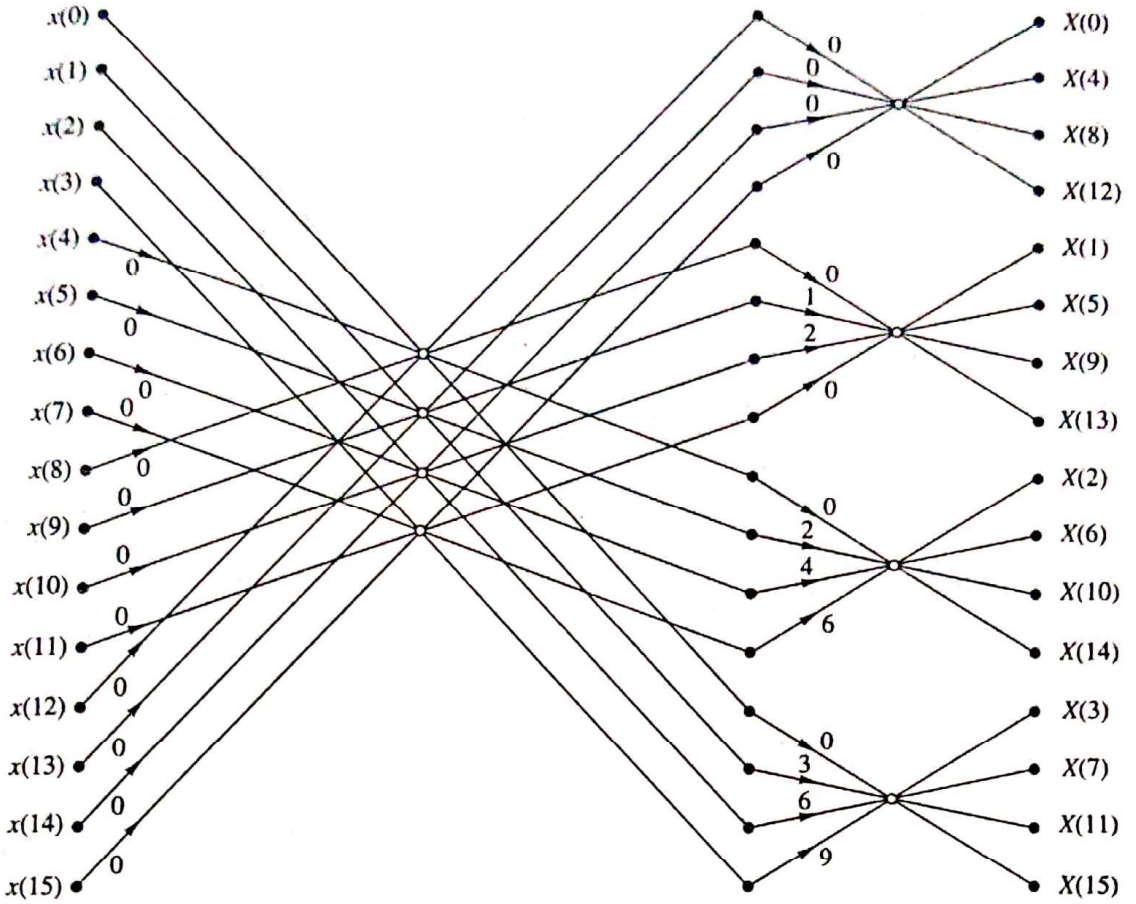

Figure 6.13 Sixteen-point radix-4 decimation-in-time algorithm with input in normal order and output in digit-reversed order.

(6.1.15) can be expressed as

$$X(p, q) = \sum_{l=0}^{(N/4)-1} G(l, q) W_{N/4}^{lp} \tag{6.1.45}$$

where

$$G(l, q) = W_N^{lq} F(l, q) \qquad \begin{matrix} q = 0, 1, 2, 3 \\ l = 0, 1, \ldots, \dfrac{N}{4} - 1 \end{matrix} \tag{6.1.46}$$

and

$$F(l, q) = \sum_{m=0}^{3} x(l, m) W_4^{mq} \qquad \begin{matrix} q = 0, 1, 2, 3 \\ l = 0, 1, 2, 3, \ldots, \dfrac{N}{4} - 1 \end{matrix} \tag{6.1.47}$$

We note that $X(p, q) = X(4p + q)$, $q = 0, 1, 2, 3$. Consequently, the N-point DFT is decimated into four $N/4$-point DFTs and hence we have a decimation-in-frequency FFT algorithm. The computations in (6.1.46) and (6.1.47) define

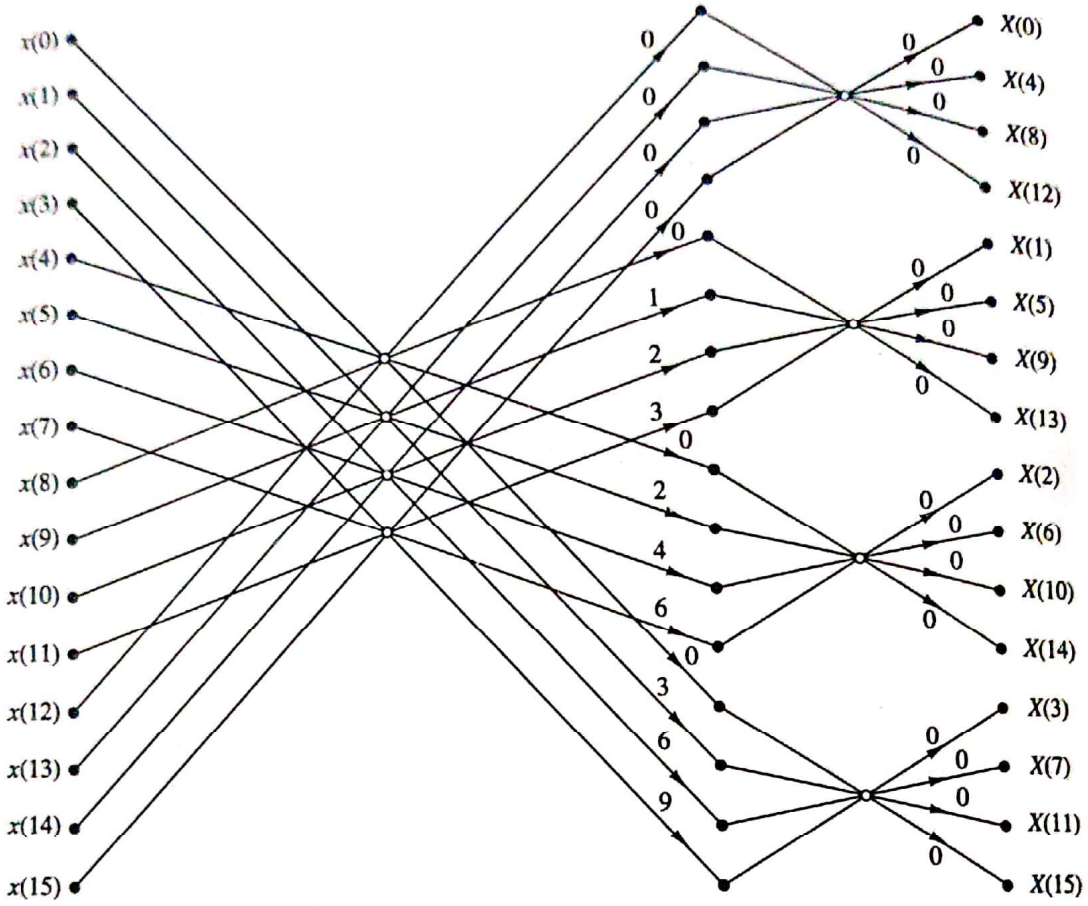

Figure 6.14 Sixteen-point, radix-4 decimation-in-frequency algorithm with input in normal order and output in digit-reversed order.

the basic radix-4 butterfly for the decimation-in-frequency algorithm. Note that the multiplications by the factors W_N^{lq} occur after the combination of the data points $x(l, m)$, just as in the case of the radix-2 decimation-in-frequency algorithm.

A 16-point radix-4 decimation-in-frequency FFT algorithm is shown in Fig. 6.14. Its input is in normal order and its output is in digit-reversed order. It has exactly the same computational complexity as the decimation-in-time radix-4 FFT algorithm.

For illustrative purposes, let us rederive the radix-4 decimation-in-frequency algorithm by breaking the N-point DFT formula into four smaller DFTs. We have

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

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

$$= \sum_{n=0}^{N/4-1} x(n) W_N^{kn} + W_N^{Nk/4} \sum_{n=0}^{N/4-1} x\left(n+\frac{N}{4}\right) W_N^{kn}$$
$$+ W_N^{kN/2} \sum_{n=0}^{N/4-1} x\left(n+\frac{N}{2}\right) W_N^{nk} + W_N^{3kN/4} \sum_{n=0}^{N/4-1} x\left(n+\frac{3N}{4}\right) W_N^{kn} \tag{6.1.48}$$

From the definition of the twiddle factors, we have

$$W_N^{kN/4} = (-j)^k \qquad W_N^{Nk/2} = (-1)^k \qquad W_N^{3Nk/4} = (j)^k \tag{6.1.49}$$

After substitution of (6.1.49) into (6.1.48), we obtain

$$X(k) = \sum_{n=0}^{N/4-1} \left[x(n) + (-j)^k x\left(n+\frac{N}{4}\right) + (-1)^k x\left(n+\frac{N}{4}\right) + (j)^k x\left(n+\frac{3N}{4}\right) \right] W_N^{nk} \tag{6.1.50}$$

The relation in (6.1.50) is not an $N/4$-point DFT because the twiddle factor depends on N and not on $N/4$. To convert it into an $N/4$-point DFT, we subdivide the DFT sequence into four $N/4$-point subsequences, $X(4k)$, $X(4k+1)$, $X(4k+2)$, and $X(4k+3)$, $k = 0, 1, \ldots, N/4-1$. Thus we obtain the radix-4 decimation-in-frequency DFT as

$$X(4k) = \sum_{n=0}^{N/4-1} \left[x(n) + x\left(n+\frac{N}{4}\right) + x\left(n+\frac{N}{2}\right) + x\left(n+\frac{3N}{4}\right) \right] W_N^0 W_{N/4}^{kn} \tag{6.1.51}$$

$$X(4k+1) = \sum_{n=0}^{N/4-1} \left[x(n) - jx\left(n+\frac{N}{4}\right) - x\left(n+\frac{N}{2}\right) + jx\left(n+\frac{3N}{4}\right) \right] W_N^n W_{N/4}^{kn} \tag{6.1.52}$$

$$X(4k+2) = \sum_{n=0}^{N/4-1} \left[x(n) - x\left(n+\frac{N}{4}\right) + x\left(n+\frac{N}{2}\right) - x\left(n+\frac{3N}{4}\right) \right] W_N^{2n} W_{N/4}^{kn} \tag{6.1.53}$$

$$X(4k+3) = \sum_{n=0}^{N/4-1} \left[x(n) + jx\left(n+\frac{N}{4}\right) - x\left(n+\frac{N}{2}\right) - jx\left(n+\frac{3N}{4}\right) \right] W_N^{3n} W_{N/4}^{kn} \tag{6.1.54}$$

where we have used the property $W_N^{4kn} = W_{N/4}^{kn}$. Note that the input to each $N/4$-point DFT is a linear combination of four signal samples scaled by a twiddle factor. This procedure is repeated ν times, where $\nu = \log_4 N$.

6.1.5 Split-Radix FFT Algorithms

An inspection of the radix-2 decimation-in-frequency flowgraph shown in Fig. 6.11 indicates that the even-numbered points of the DFT can be computed independently of the odd-numbered points. This suggests the possibility of using different computational methods for independent parts of the algorithm with the objective of reducing the number of computations. The split-radix FFT (SRFFT) algorithms exploit this idea by using both a radix-2 and a radix-4 decomposition in the same FFT algorithm.

We illustrate this approach with a decimation-in-frequency SRFFT algorithm due to Duhamel (1986). First, we recall that in the radix-2 decimation-in-frequency FFT algorithm, the even-numbered samples of the N-point DFT are given as

$$X(2k) = \sum_{n=0}^{N/2-1} \left[x(n) + x\left(n + \frac{N}{2}\right) \right] W_{N/2}^{nk} \qquad k = 0, 1, \ldots, \frac{N}{2} - 1 \tag{6.1.55}$$

Note that these DFT points can be obtained from an $N/2$-point DFT without any additional multiplications. Consequently, a radix-2 suffices for this computation.

The odd-numbered samples $\{X(2k+1)\}$ of the DFT require the premultiplication of the input sequence with the twiddle factors W_N^n. For these samples a radix-4 decomposition produces some computational efficiency because the four-point DFT has the largest multiplication-free butterfly. Indeed, it can be shown that using a radix greater than 4, does not result in a significant reduction in computational complexity.

If we use a radix-4 decimation-in-frequency FFT algorithm for the odd-numbered samples of the N-point DFT, we obtain the following $N/4$-point DFTs:

$$\begin{aligned} X(4k+1) = \sum_{n=0}^{N/4-1} & \{[x(n) - x(n+N/2)] \\ & - j[x(n+N/4) - x(n+3N/4)]\} W_N^n W_{N/4}^{kn} \end{aligned} \tag{6.1.56}$$

$$\begin{aligned} X(4k+3) = \sum_{n=0}^{N/4-1} & \{[x(n) - x(n+N/2)] \\ & + j[x(n+N/4) - x(n+3N/4)]\} W_N^{3n} W_{N/4}^{kn} \end{aligned} \tag{6.1.57}$$

Thus the N-point DFT is decomposed into one $N/2$-point DFT without additional twiddle factors and two $N/4$-point DFTs with twiddle factors. The N-point DFT is obtained by successive use of these decompositions up to the last stage. Thus we obtain a decimation-in-frequency SRFFT algorithm.

Figure 6.15 shows the flow graph for an in-place 32-point decimation-in-frequency SRFFT algorithm. At stage A of the computation for $N = 32$, the

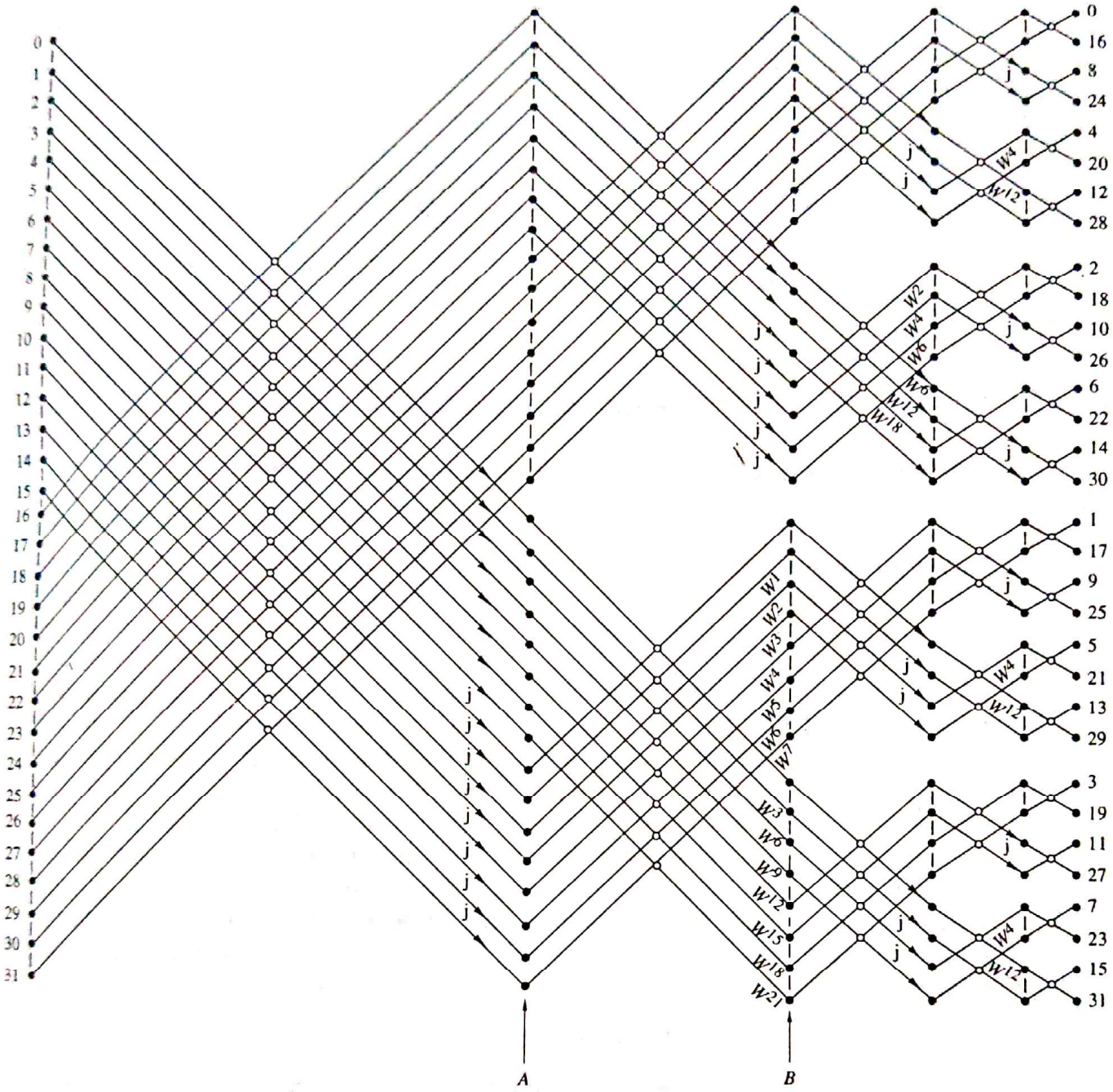

Figure 6.15 Length 32 split-radix FFT algorithms from paper by Duhamel (1986); reprinted with permission from the IEEE.

top 16 points constitute the sequence

$$g_0(n) = x(n) + x(n + N/2) \qquad 0 \le n \le 15 \tag{6.1.58}$$

This is the sequence required for the computation of $X(2k)$. The next 8 points constitute the sequence

$$g_1(n) = x(n) - x(n + N/2) \qquad 0 \le n \le 7 \tag{6.1.59}$$

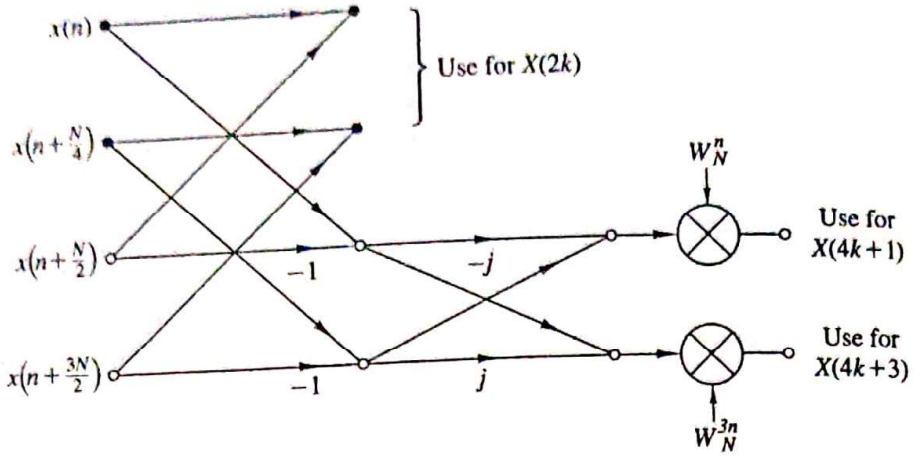

Figure 6.16 Butterfly for SRFFT algorithm.

The bottom eight points constitute the sequence $jg_2(n)$, where

$$g_2(n) = x(n + N/4) - x(n + 3N/4) \qquad 0 \le n \le 7 \tag{6.1.60}$$

The sequences $g_1(n)$ and $g_2(n)$ are used in the computation of $X(4k + 1)$ and $X(4k+3)$. Thus, at stage A we have completed the first decimation for the radix-2 component of the algorithm. At stage B, the bottom eight points constitute the computation of $[g_1(n)+jg_2(n)]W_{32}^{3n}$, $0 \le n \le 7$, which is used to compute $X(4k+3)$, $0 \le k \le 7$. The next eight points from the bottom constitute the computation of $[g_1(n) - jg_2(n)]\ W_{32}^{n}$, $0 \le n \le 7$, which is used to compute $X(4k + 1)$, $0 \le k \le 7$. Thus at stage B, we have completed the first decimation for the radix-4 algorithm, which results in two 8-point sequences. Hence the basic butterfly computation for the SRFFT algorithm has the "L-shaped" form illustrated in Fig. 6.16.

Now we repeat the steps in the computation above. Beginning with the top 16 points at stage A, we repeat the decomposition for the 16-point DFT. In other words, we decompose the computation into an eight-point, radix-2 DFT and two four-point, radix-4 DFTs. Thus at stage B, the top eight points constitute the sequence (with $N = 16$)

$$g_0'(n) = g_0(n) + g_0(n + N/2) \qquad 0 \le n \le 7 \tag{6.1.61}$$

and the next eight points constitute the two four-point sequences $g_1'(n)$ and $jg_2'(n)$, where

$$\begin{aligned} g_1'(n) &= g_0(n) - g_0(n + N/2) & 0 \le n \le 3 \\ g_2'(n) &= g_0(n + N/4) - g_0(n + 3N/4) & 0 \le n \le 3 \end{aligned} \tag{6.1.62}$$

The bottom 16 points of stage B are in the form of two eight-point DFTs. Hence each eight-point DFT is decomposed into a four-point, radix-2 DFT and a four-point, radix-4 DFT. In the final stage, the computations involve the combination of two-point sequences.

Table 6.2 presents a comparison of the number of *nontrivial* real multiplications and additions required to perform an N-point DFT with complex-valued

TABLE 6.2 NUMBER OF NONTRIVIAL REAL MULTIPLICATIONS AND ADDITIONS TO COMPUTE AN N-POINT COMPLEX DFT

	Real Multiplications				Real Additions			
N	Radix 2	Radix 4	Radix 8	Split Radix	Radix 2	Radix 4	Radix 8	Split Radix
16	24	20		20	152	148		148
32	88			68	408			388
64	264	208	204	196	1,032	976	972	964
128	712			516	2,504			2,308
256	1,800	1,392		1,284	5,896	5,488		5,380
512	4,360		3,204	3,076	13,566		12,420	12,292
1,024	10,248	7,856		7,172	30,728	28,336		27,652

Source: Extracted from Duhamel (1986).

data, using a radix-2, radix-4, radix-8, and a split-radix FFT. Note that the SRFFT algorithm requires the lowest number of multiplication and additions. For this reason, it is preferable in many practical applications.

Another type of SRFFT algorithm has been developed by Price (1990). Its relation to Duhamel's algorithm described previously can be seen by noting that the radix-4 DFT terms $X(4k+1)$ and $X(4k+3)$ involve the $N/4$-point DFTs of the sequences $[g_1(n) - jg_2(n)]W_N^n$ and $[g_1(n) + jg_2(n)]W_N^{3n}$, respectively. In effect, the sequences $g_1(n)$ and $g_2(n)$ are multiplied by the factor (vector) $(1, -j) = (1, W_{32}^8)$ and by W_N^n for the computation of $X(4k+1)$, while the computation of $X(4k+3)$ involves the factor $(1, j) = (1, W_{32}^{-8})$ and W_N^{3n}. Instead, one can rearrange the computation so that the factor for $X(4k+3)$ is $(-j, -1) = -(W_{32}^{-8}, 1)$. As a result of this phase rotation, the twiddle factors in the computation of $X(4k+3)$ become exactly the same as those for $X(4k+1)$, except that they occur in mirror image order. For example, at stage B of Fig. 6.15, the twiddle factors $W^{21}, W^{18}, \ldots, W^3$ are replaced by $W^1, W^2, \ldots, W^7$, respectively. This mirror-image symmetry occurs at every subsequent stage of the algorithm. As a consequence, the number of twiddle factors that must be computed and stored is reduced by a factor of 2 in comparison to Duhamel's algorithm. The resulting algorithm is called the "mirror" FFT (MFFT) algorithm.

An additional factor-of-2 savings in storage of twiddle factors can be obtained by introducing a 90° phase offset at the midpoint of each twiddle array, which can be removed if necessary at the output of the SRFFT computation. The incorporation of this improvement into the SRFFT (or the MFFT) results in another algorithm, also due to Price (1990), called the "phase" FFT (PFFT) algorithm.

6.1.6 Implementation of FFT Algorithms

Now that we have described the basic radix-2 and radix-4 FFT algorithms, let us consider some of the implementation issues. Our remarks apply directly to

radix-2 algorithms, although similar comments may be made about radix-4 and higher-radix algorithms.

Basically, the radix-2 FFT algorithm consists of taking two data points at a time from memory, performing the butterfly computations and returning the resulting numbers to memory. This procedure is repeated many times ($(N \log_2 N)/2$ times) in the computation of an N-point DFT.

The butterfly computations require the twiddle factors $\{W_N^k\}$ at various stages in either natural or bit-reversed order. In an efficient implementation of the algorithm, the phase factors are computed once and stored in a table, either in normal order or in bit-reversed order, depending on the specific implementation of the algorithm.

Memory requirement is another factor that must be considered. If the computations are performed in place, the number of memory locations required is $2N$ since the numbers are complex. However, we can instead double the memory to 4N, thus simplifying the indexing and control operations in the FFT algorithms. In this case we simply alternate in the use of the two sets of memory locations from one stage of the FFT algorithm to the other. Doubling of the memory also allows us to have both the input sequence and the output sequence in normal order.

There are a number of other implementation issues regarding indexing, bit reversal, and the degree of parallelism in the computations. To a large extent, these issues are a function of the specific algorithm and the type of implementation, namely, a hardware or software implementation. In implementations based on a fixed-point arithmetic, or floating-point arithmetic on small machines, there is also the issue of round-off errors in the computation. This topic is considered in Section 6.4.

Although the FFT algorithms described previously were presented in the context of computing the DFT efficiently, they can also be used to compute the IDFT, which is

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-nk} \tag{6.1.63}$$

The only difference between the two transforms is the normalization factor $1/N$ and the sign of the phase factor W_N. Consequently, an FFT algorithm for computing the DFT, can be converted to an FFT algorithm for computing the IDFT by changing the sign on all the phase factors and dividing the final output of the algorithm by N.

In fact, if we take the decimation-in-time algorithm that we described in Section 6.1.3, reverse the direction of the flow graph, change the sign on the phase factors, interchange the output and input, and finally, divide the output by N, we obtain a decimation-in-frequency FFT algorithm for computing the IDFT. On the other hand, if we begin with the decimation-in-frequency FFT algorithm described in Section 6.1.3 and repeat the changes described above, we obtain a decimation-in-time FFT algorithm for computing the IDFT. Thus it is a simple matter to devise FFT algorithms for computing the IDFT.

Finally, we note that the emphasis in our discussion of FFT algorithms was on radix-2, radix-4, and split-radix algorithms. These are by far the most widely used in practice. When the number of data points is not a power of 2 or 4, it is a simple matter to pad the sequence $x(n)$ with zeros such that $N = 2^\nu$ or $N = 4^\nu$.

The measure of complexity for FFT algorithms that we have emphasized is the required number of arithmetic operations (multiplications and additions). Although this is a very important benchmark for computational complexity, there are other issues to be considered in practical implementation of FFT algorithms. These include the architecture of the processor, the available instruction set, the data structures for storing twiddle factors, and other considerations.

For general-purpose computers, where the cost of the numerical operations dominate, radix-2, radix-4, and split-radix FFT algorithms are good candidates. However, in the case of special-purpose digital signal processors, featuring single-cycle multiply-and-accumulate operation, bit-reversed addressing, and a high degree of instruction parallelism, the structural regularity of the algorithm is equally important as arithmetic complexity. Hence for DSP processors, radix-2 or radix-4 decimation-in-frequency FFT algorithms are preferable in terms of speed and accuracy. The irregular structure of the SRFFT may render it less suitable for implementation on digital signal processors. Structural regularity is also important in the implementation of FFT algorithms on vector processors, multiprocessors, and in VLSI. Interprocessor communication is an important consideration in such implementations on parallel processors.

In conclusion, we have presented several important considerations in the implementation of FFT algorithms. Advances in digital signal processing technology, in hardware and software, will continue to influence the choice among FFT algorithms for various practical applications.

APPLICATIONS OF FFT ALGORITHMS

The FFT algorithms described in the preceding section find application in a variety of areas, including linear filtering, correlation, and spectrum analysis. Basically, the FFT algorithm is used as an efficient means to compute the DFT and the IDFT.

In this section we consider the use of the FFT algorithm in linear filtering and in the computation of the crosscorrelation of two sequences. The use of the FFT in spectrum analysis is considered in Chapter 12. In addition we illustrate how to enhance the efficiency of the FFT algorithm by forming complex-valued sequences from real-valued sequences prior to the computation of the DFT.