Fast Fourier Transform (FFT)

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.

From this point, we change the notation that *X(k)*, instead of *y(k)* in previous sections, represents the Fourier coefficients of *x(n)*.

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

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

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

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

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

Since 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 :

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

$$\text{Periodicit y property}: W_N^{k+N} = W_N^k$$

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

Radix-2 FFT Algorithms

Let us consider the computation of the $N = 2^v$ point DFT by the divide-and conquer approach. We split 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,

$$f_1(n) = x(2n)$$

$$f_2(n) = x(2n+1), \qquad n = 0,1,\ldots,\frac{N}{2}-1$$

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 DFT's of the decimated sequences as follows:

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

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

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

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 the equation may be expressed as

$$\begin{aligned} X(k) &= F_1(k) + W_N^k F_2(k), \qquad k = 0,1,\ldots,\frac{N}{2}-1 \\ X(k+\frac{N}{2}) &= F_1(k) - W_N^k F_2(k), \qquad k = 0,1,\ldots,\frac{N}{2}-1 \end{aligned}$$

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.

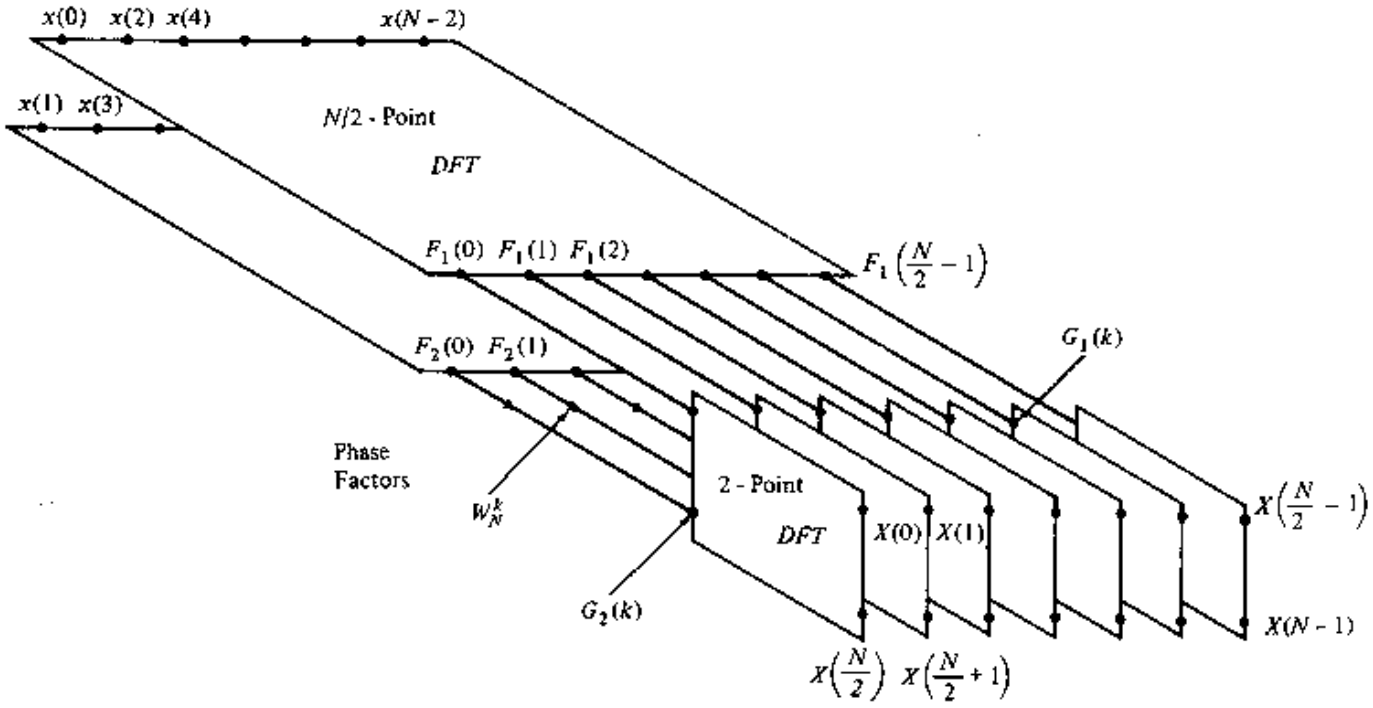


Figure TC.3.1 First step in the decimation-in-time algorithm.

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

$$F_1(k) = \mathrm{F}\{f_1(2n)\} + W_{N/2}^k \mathrm{F}\{f_1(2n+1)\}, \quad k = 0,1,\ldots,\frac{N}{4}-1; \quad n = 0,1,\ldots,\frac{N}{4}-1$$

$$F_1\left(k+\frac{N}{4}\right) = \mathrm{F}\{f_1(2n)\} - W_{N/2}^k \mathrm{F}\{f_1(2n+1)\}, \quad k = 0,1,\ldots,\frac{N}{4}-1; \quad n = 0,1,\ldots,\frac{N}{4}-1$$

$$F_2(k) = \mathrm{F}\{f_2(2n)\} + W_{N/2}^k \mathrm{F}\{f_2(2n+1)\}, \quad k = 0,1,\ldots,\frac{N}{4}-1; \quad n = 0,1,\ldots,\frac{N}{4}-1$$

$$F_2\left(k+\frac{N}{4}\right) = \mathrm{F}\{f_2(2n)\} - W_{N/2}^k \mathrm{F}\{f_2(2n+1)\}, \quad k = 0,1,\ldots,\frac{N}{4}-1; \quad n = 0,1,\ldots,\frac{N}{4}-1$$

$\mathrm{F}\{*\}$ represents Fourier tr ansform

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 $N\log_2 N$.

For illustrative purposes, Figure TC.3.2 depicts the computation of $N = 8$ point DFT. We observe that the computation is performed in tree stages, beginning with the computations of four two-point DFTs, then two four-point DFTs, and finally, one eight-point DFT. The combination for the smaller DFTs to form the larger DFT is illustrated in Figure TC.3.3 for $N = 8$.

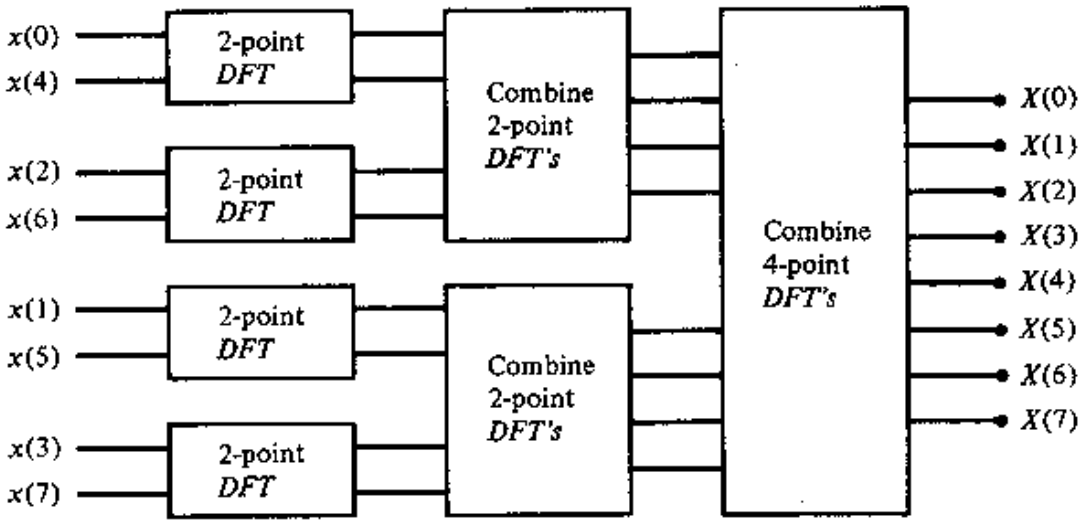


Figure TC.3.2 Three stages in the computation of an $N = 8$-point DFT.

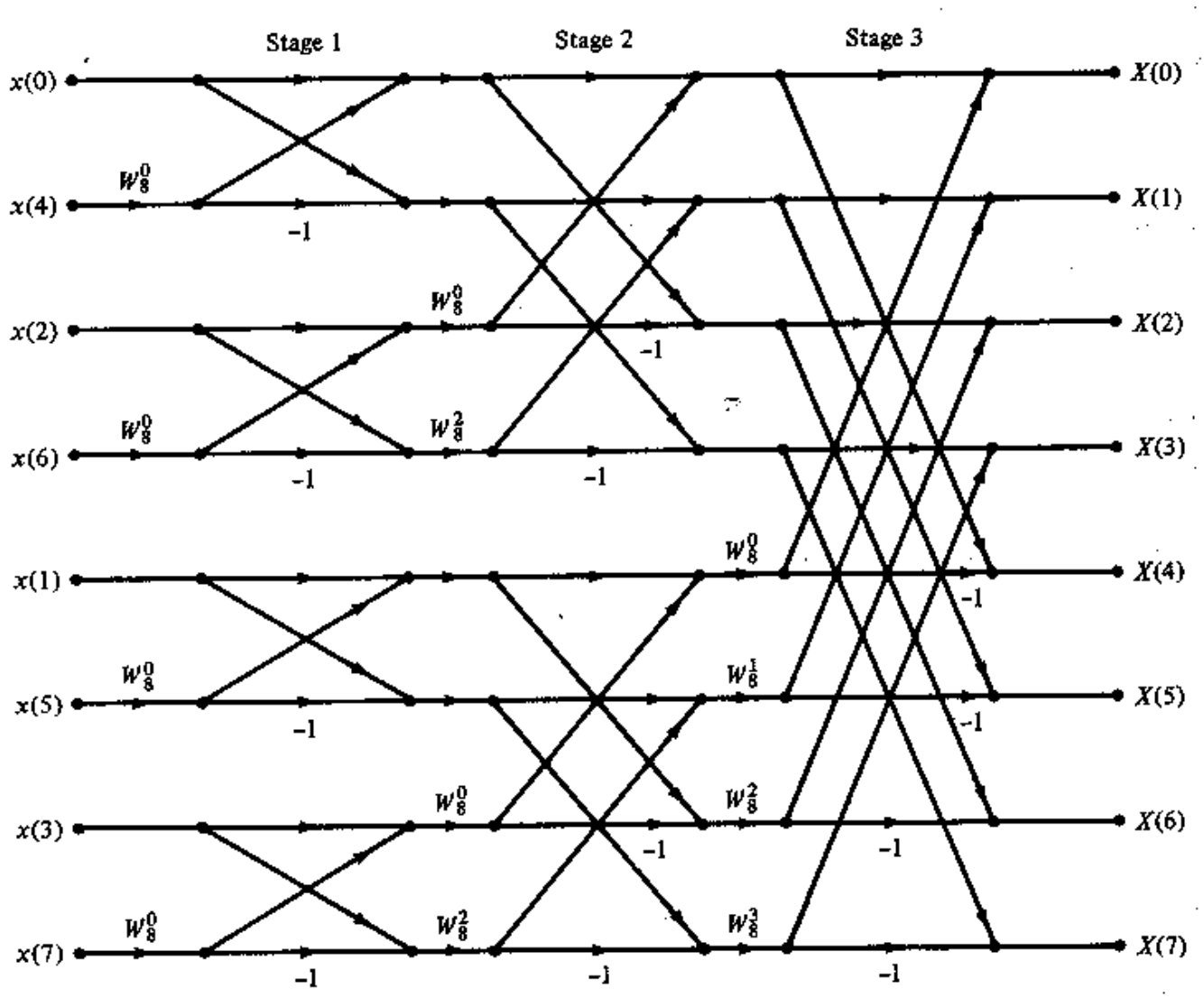


Figure TC.3.3 Eight-point decimation-in-time FFT algorithm.

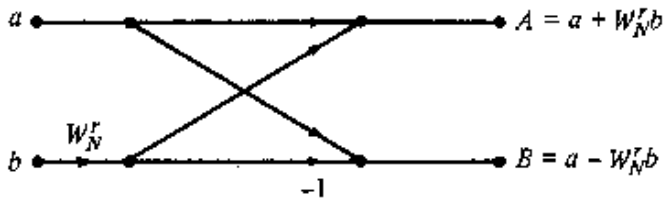
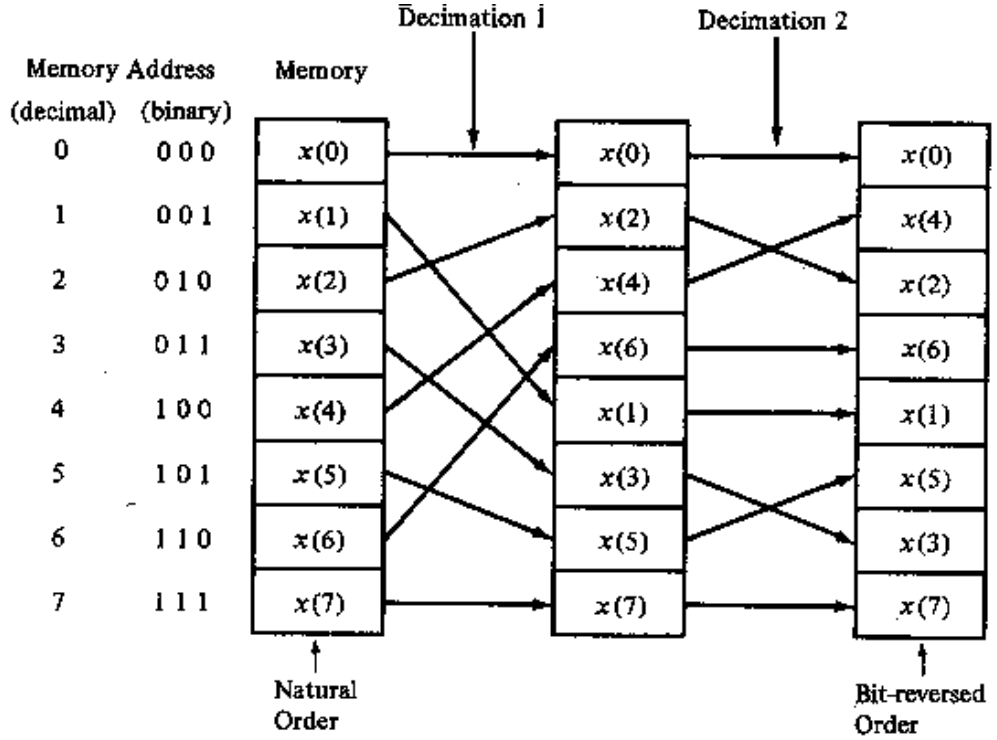


Figure TC.3.4 Basic butterfly computation in the decimation-in-time FFT algorithm.

An important observation is concerned with the order of the input data sequence after it is decimated (v-1) times. For example, if we consider the case where N = 8, we know that the first decimation yeilds 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 Figure TC.3.5, which illustrates the decimation of the eight-point sequence.



(a)

$(n_2 n_1 n_0)$	→	$(n_0 n_2 n_1)$	→	$(n_0 n_1 n_2)$
(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 TC.3.5 Shuffling of the data and bit reversal.

Another important radix-2 FFT algorithm, called the decimation-in-frequency algorithm, is obtained by using the divide-and-conquer approach. 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

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

Since $W_N^{kN/2} = (-1)^k$

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

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

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

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

The computational procedure above can be repeated through decimation of the $N/2$-point DFTs $X(2k)$ and $X(2k+1)$. The entire process involves $v = \log_2 N$ stages of decimation, where each stage involves $N/2$ butterflies of the type shown in Figure TC.3.7. Consequently, the computation of the N-point DFT via the decimation-in-frequency FFT 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 Figure TC.3.8.

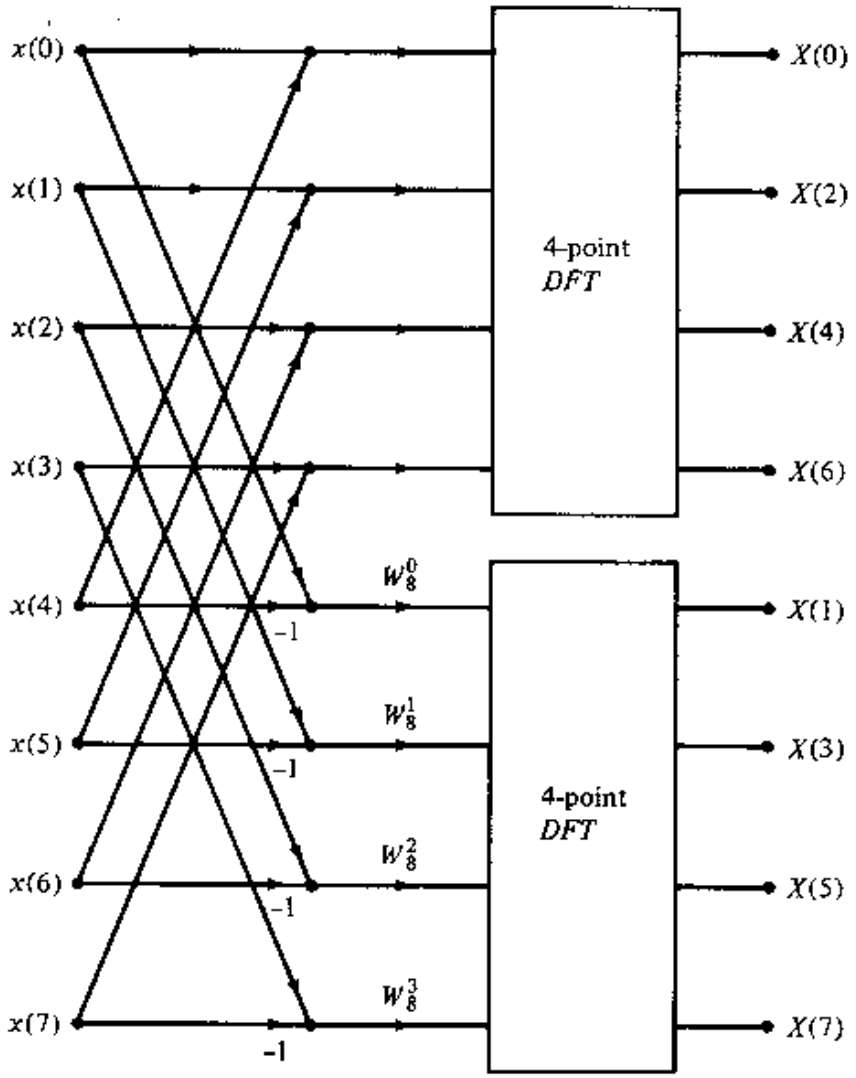


Figure TC.3.6 First stage of the decimation-in-frequency FFT algorithm.

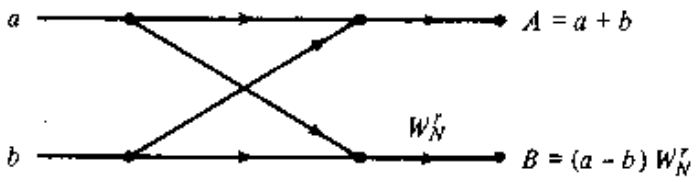


Figure TC.3.7 Basic butterfly computation in the decimation-in-frequency.

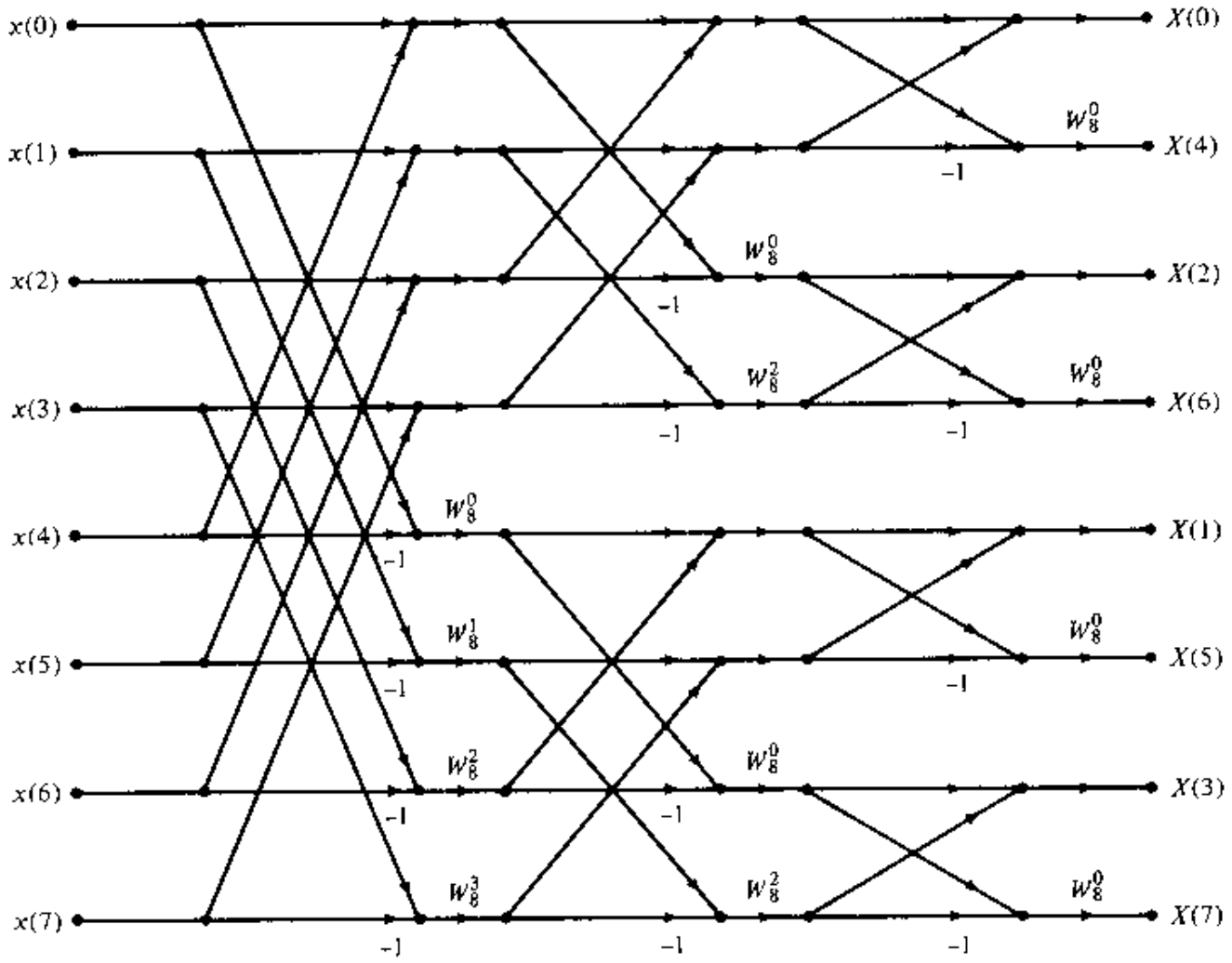


Figure TC.3.8 $N = 8$-piont decimation-in-frequency FFT algorithm.

We observe from Figure TC.3.8 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.

Radix-4 FFT Algorithm

When the number of data points N in the DFT is a power of 4 (i.e., $N = 4^v$), 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-r FFT algorithm.

Let us begin by describing a radix-4 decimation-in-time FFT algorithm briefly. We split or decimate the N-point input sequence into four subsequences, $x(4n)$, $x(4n+1)$, $x(4n+2)$, $x(4n+3)$, $n = 0, 1, \ldots, N/4-1$.

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

$$F(l,q) = \sum_{m=0}^{(N/4)-1} x(l,m) W_{N/4}^{mq}$$

$$p = 0,1,2,3; \quad l = 0,1,2,3; \quad q = 0,1,2,\ldots,\frac{N}{4}-1$$

and

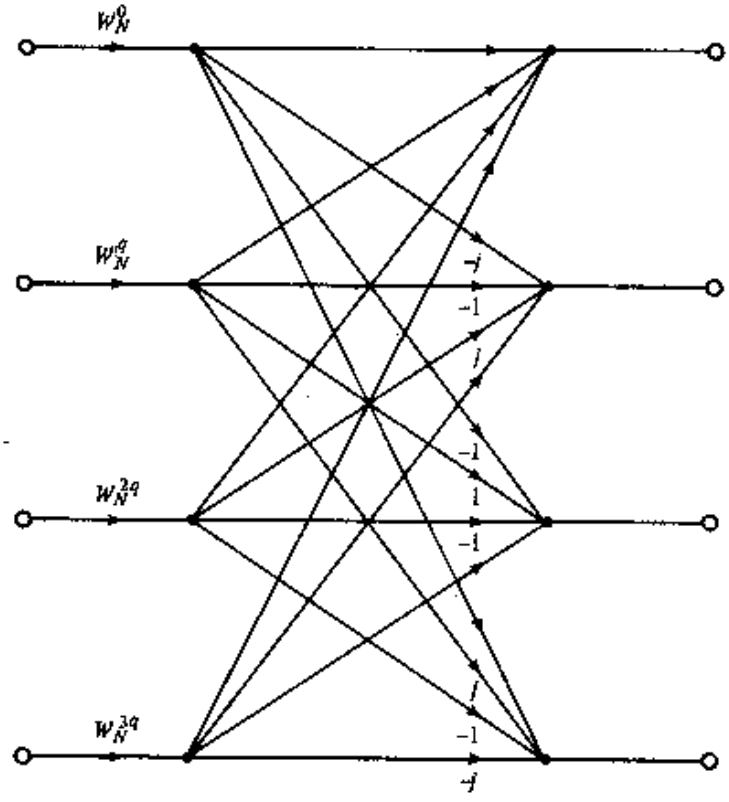
$$x(l,m) = x(4m+1)$$

$$X(p,q) = X(\frac{N}{4}p + q)$$

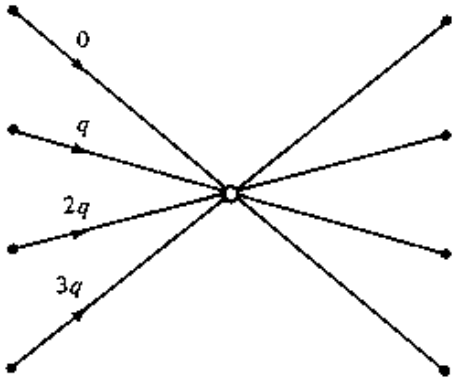
Thus the four $N/4$-point DFTs $F(l, q)$obtained from the above equation are combined to yield the N-point DFT. The expression for combining the $N/4$-point DFTs defines a radix-4 decimation-in-time butterfly, which can be expressed in matrix form as

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

The radix-4 butterfly is depicted in Figure TC.3.9a and in a more compact form in Figure TC.3.9b. Note that each butterfly involves three complex multiplications, since $W_N^0 = 1$, and 12 complex additions.



(a)



(b)

Figure TC.3.9 Basic butterfly computation in a radix-4 FFT algorithm.

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 ty expressing the matrix of the linear transformation mentioned previously as a product of two matrices as follows:

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

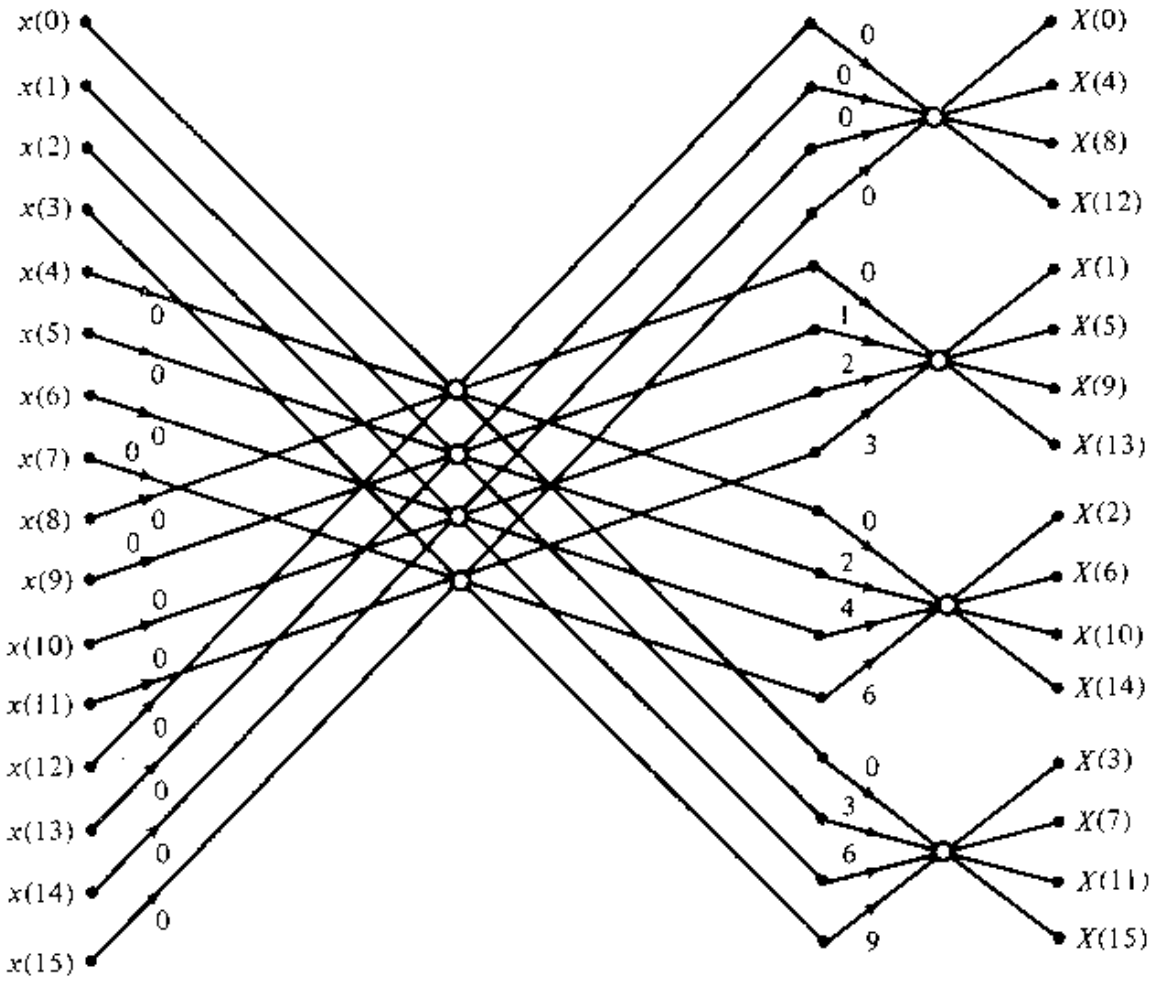


Figure TC.3.10 Sixteen-point radix-4 decimation-in-time algorithm with input in normal order and output in digit-reversed order

A 16-point, radix-4 decimation-in-frequency FFT algorithm is shown in Figure TC.3.11. 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 radex-4 FFT algorithm.

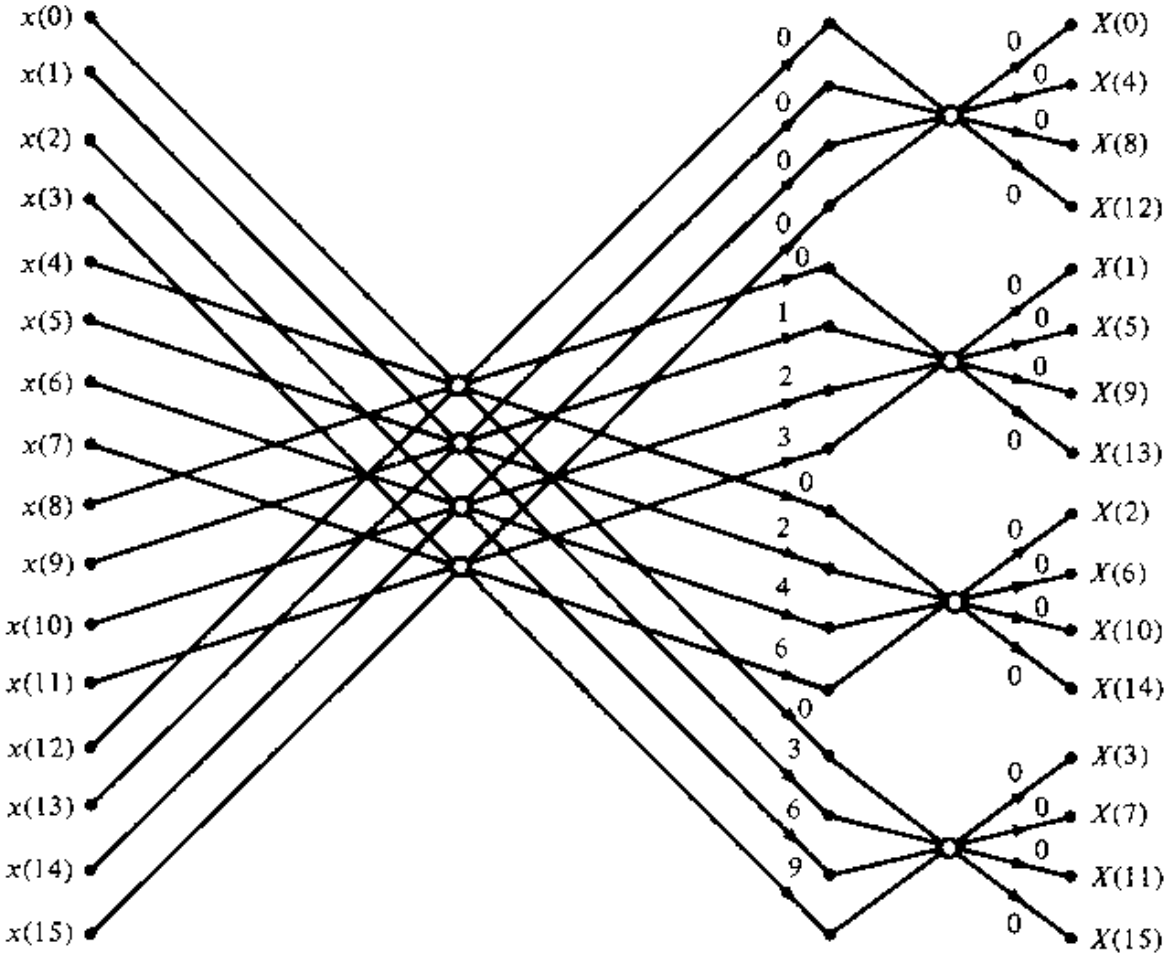


Figure TC.3.11 Sixteen-point, radix-4 decimation-in-frequency algorithm with input in normal order and output in digit-reversed order.

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

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) W_N^{kn} \\ &= \sum_{n=0}^{N/4-1} x(n) W_N^{kn} + \sum_{n=N/4}^{N/2-1} x(n) W_N^{kn} + \sum_{n=N/2}^{3N/4-1} x(n) W_N^{kn} + \sum_{n=3N/4}^{N-1} x(n) W_N^{kn} \\ &= \sum_{n=0}^{N/4-1} x(n) W_N^{kn} + W_N^{Nk/4} \sum_{n=0}^{N/4-1} x\left(n+\frac{N}{4}\right) W_N^{kn} + \\ &\quad W_N^{Nk/2} \sum_{n=0}^{N/4-1} x\left(n+\frac{N}{2}\right) W_N^{kn} + W_N^{3Nk/4} \sum_{n=0}^{N/4-1} x\left(n+\frac{3N}{4}\right) W_N^{kn} \end{aligned}$$

From the definition of the twiddle factors, we have

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

Thus

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

The relation 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 subdivede the DFT sequence into four $N/4$-point subsequences, $X(4k)$, $X(4k+1)$, $X(4k+2)$, and $X(4k+3)$, $k = 0, 1, ..., N/4$. Thus we obtain the radix-4 decimation-in frequency DFT as

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

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

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

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

where we have used the property $W_N^{4kn} = W^{kn}_{N/4}$. 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 v times, where $v = \log_4 N$.

Split-Radix FFT Algorithms

An inspection of the radix-2 decimation-in-frequency flowgraph shown in Figure TC.3.8 indicates that the even-numbered pints of the DFT can be computed independently of the odd-numbered points. This suggests teh 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.

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

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

A radix-2 suffices for this computation.

The odd-numbered samples $\{X(2k+1)\}$ of the DFT require the pre-multiplication 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:

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

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

Figure TC.3.12 shows the flow graph for an in-place 32-point decimation-in-frequency SFFT algorithm.

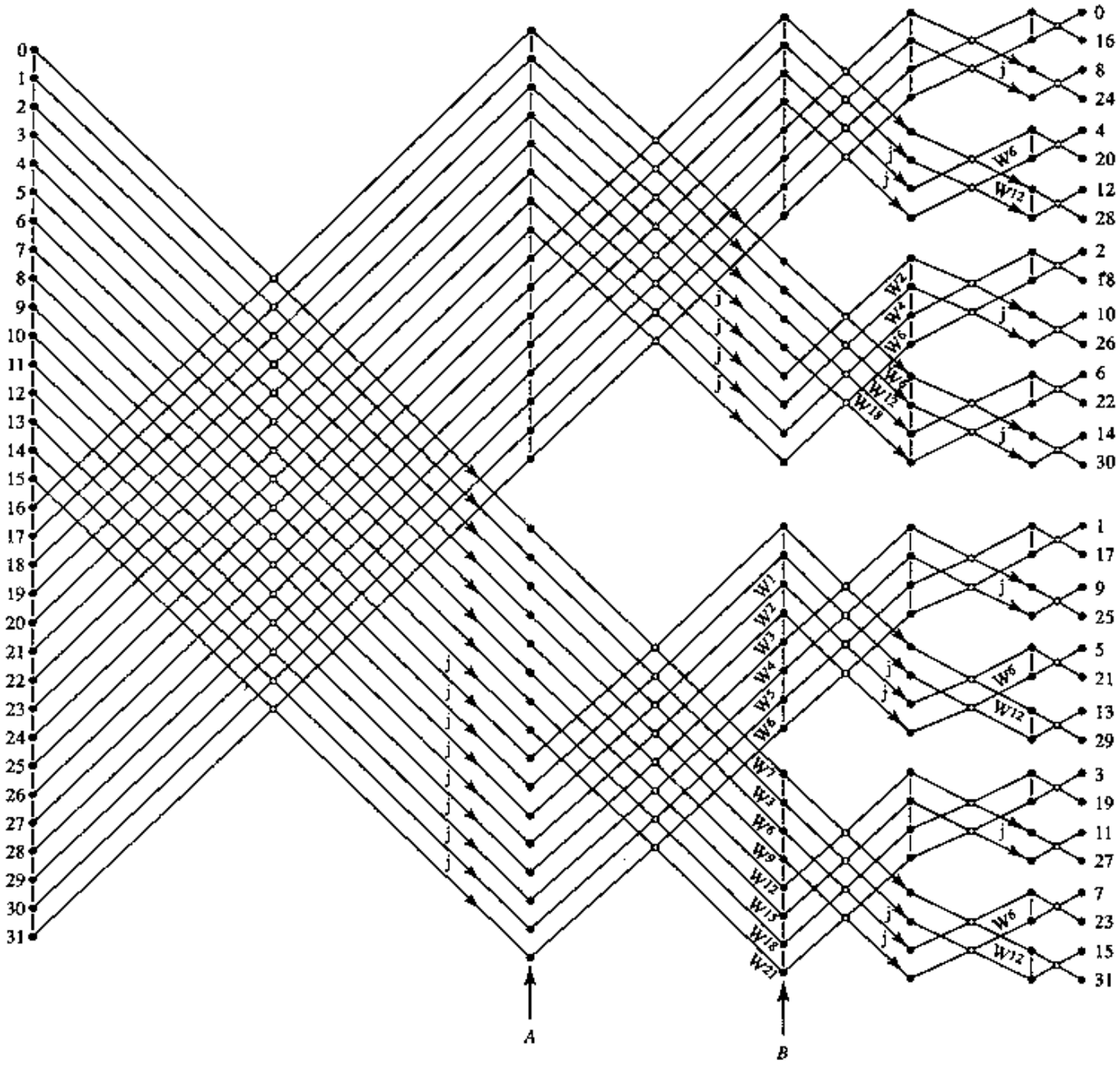


Figure TC.3.12 Length 23 split-radix FFT algorithms from paper by Duhamel (1986); reprinted with permission from the IEEE

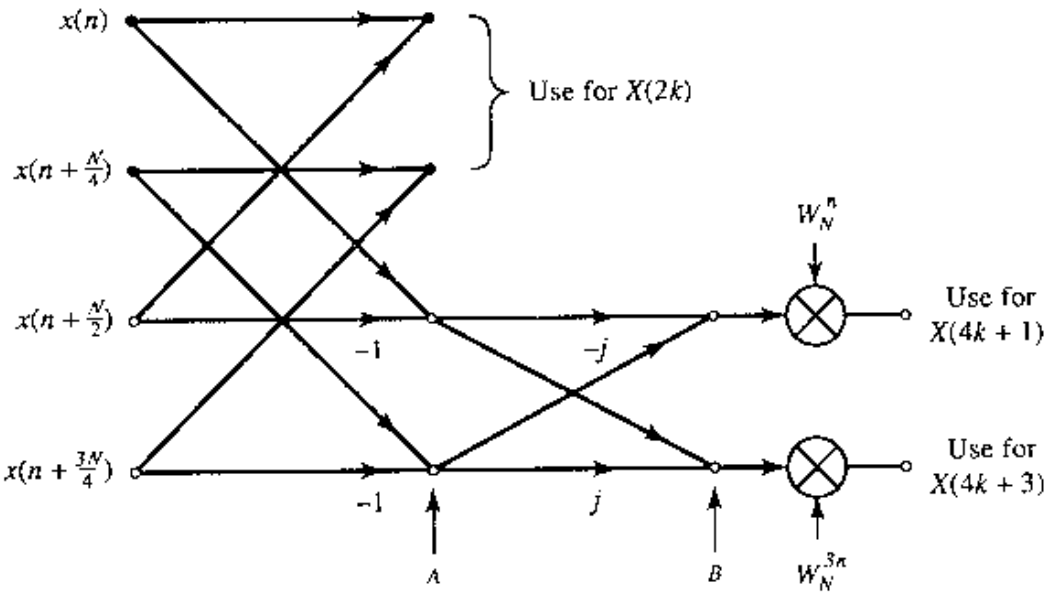


Figure TC.3.13 Butterfly for SRFFT algorithm.

	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	1032	976	972	964
128	72			516	2054			2308
256	1800	1392		1284	5896	5488		5380
512	4360		3204	3076	13566		12420	12292
1024	10248	7856		7172	30728	28336		27652

Table TC.3.1 Number of Nontrivial Real Multiplcations and Additions to Compute an *N*-point Complex DFT