Unit -2

FAST FOURIER TRANSFORM

EFFICIENT COMPUTATION OF DFT:

In this section we represent several methods for computing 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 considerably attention by many mathematicians, engineers and scientists. Basically the computation is done using the formula method.

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

where

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

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

Similarly, the IDFT becomes

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

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.

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 $x(n)$, $0 \leq n \leq N-1$, and the phase factors and to store the results. The variety of DFT algorithms optimize each of these computational processes in a different way.

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 DFT. This basic approach leads to a family o f computationally efficient algorithm s know n collectively as FFT algorithms.
T o 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, $N = L\ M$

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.

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.

6.1.3 Radix-2 FFT Algorithms

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:

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) W_N^{kn} \qquad k = 0, 1, \ldots, N-1 \\ &= \sum_{n \text{ even}} x(n) W_N^{kn} + \sum_{n \text{ odd}} x(n) W_N^{kn} \\ &= \sum_{m=0}^{(N/2)-1} x(2m) W_N^{2mk} + \sum_{m=0}^{(N/2)-1} x(2m+1) W_N^{k(2m+1)} \end{aligned} \tag{6.1.24}$$

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

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

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

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

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

$N/4$-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 $\{v_{ij}(n)\}$.

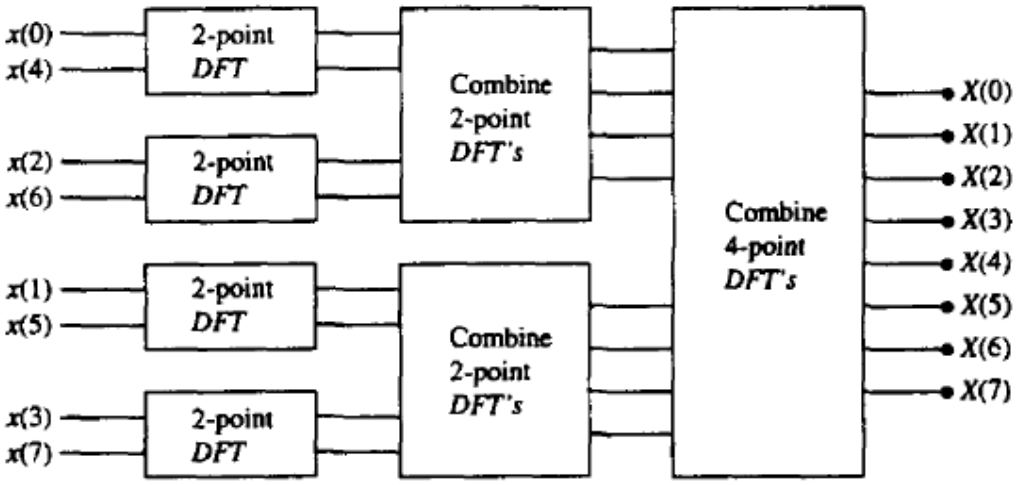


Figure 6.5 Three stages in the computation of an $N = 8$-point DFT.

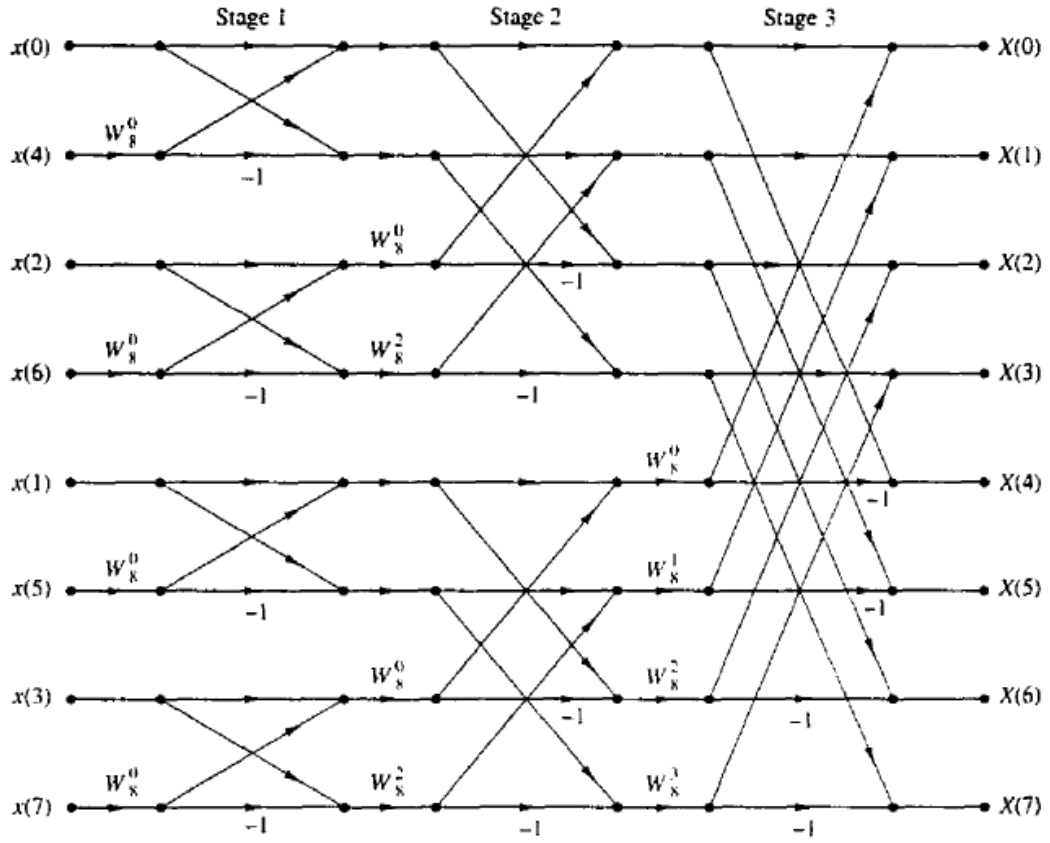


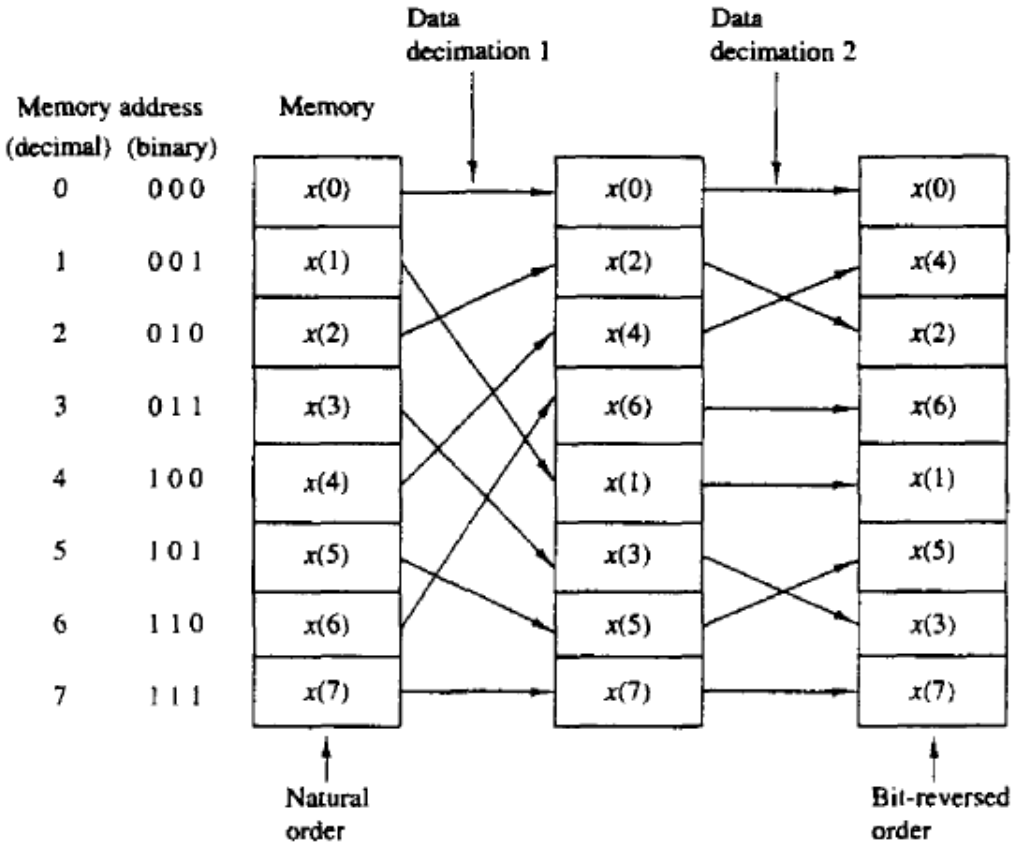
Figure 6.6 Eight-point decimation-in-time FFT algorithm.

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)

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 $W_N^2 = W_{N/2}$.

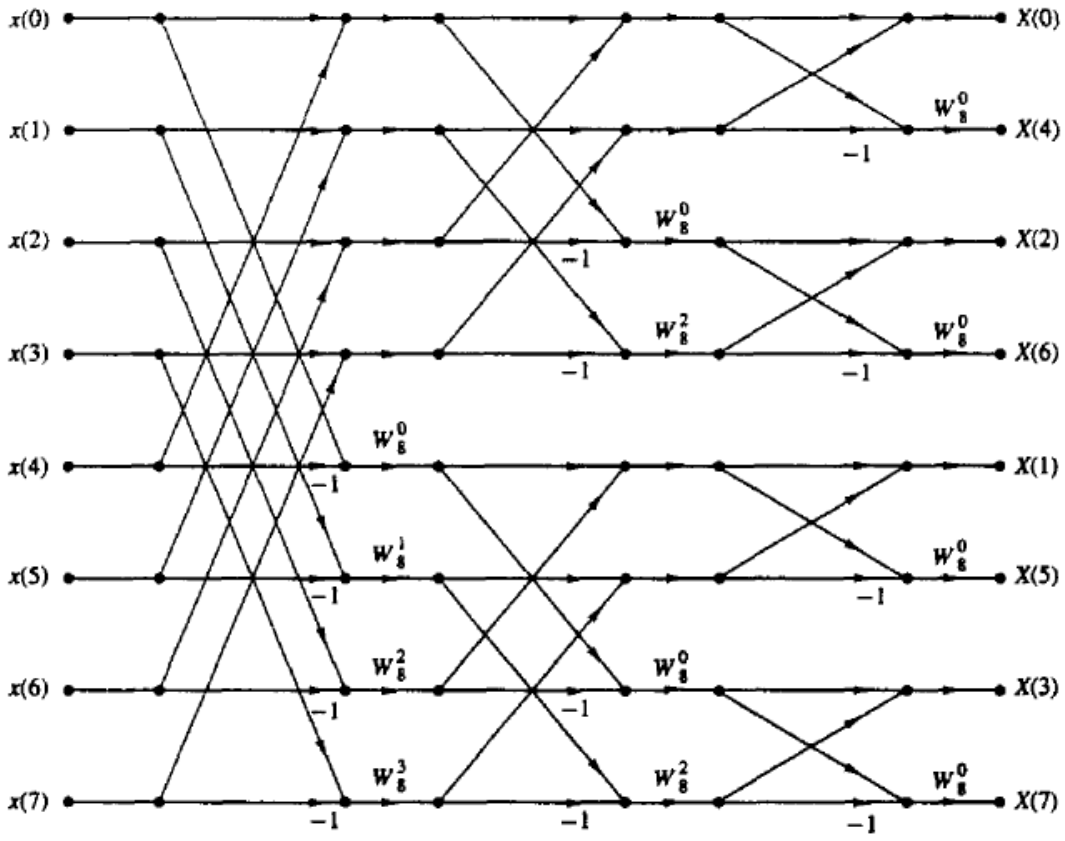
If we define the $N/2$-point sequences $g_1(n)$ and $g_2(n)$ as

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

then

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

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.


x(0)
x(1)
x(2)
x(3)
x(4)
x(5)
x(6)
x(7)
X(0)
X(4)
X(2)
X(6)
X(1)
X(5)
X(3)
X(7)
W_8^0
W_8^1
W_8^2
W_8^3
−1

6.1.4 Radix-4 FFT Algorithms

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

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

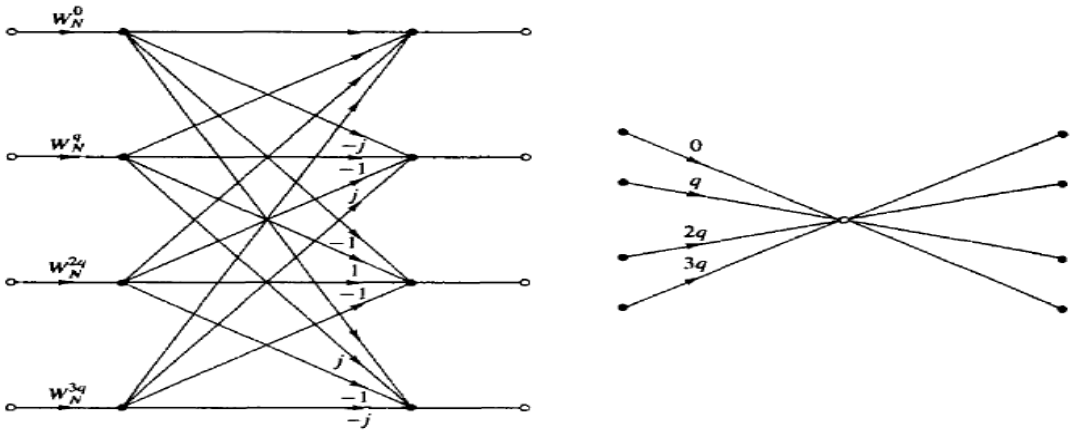
and

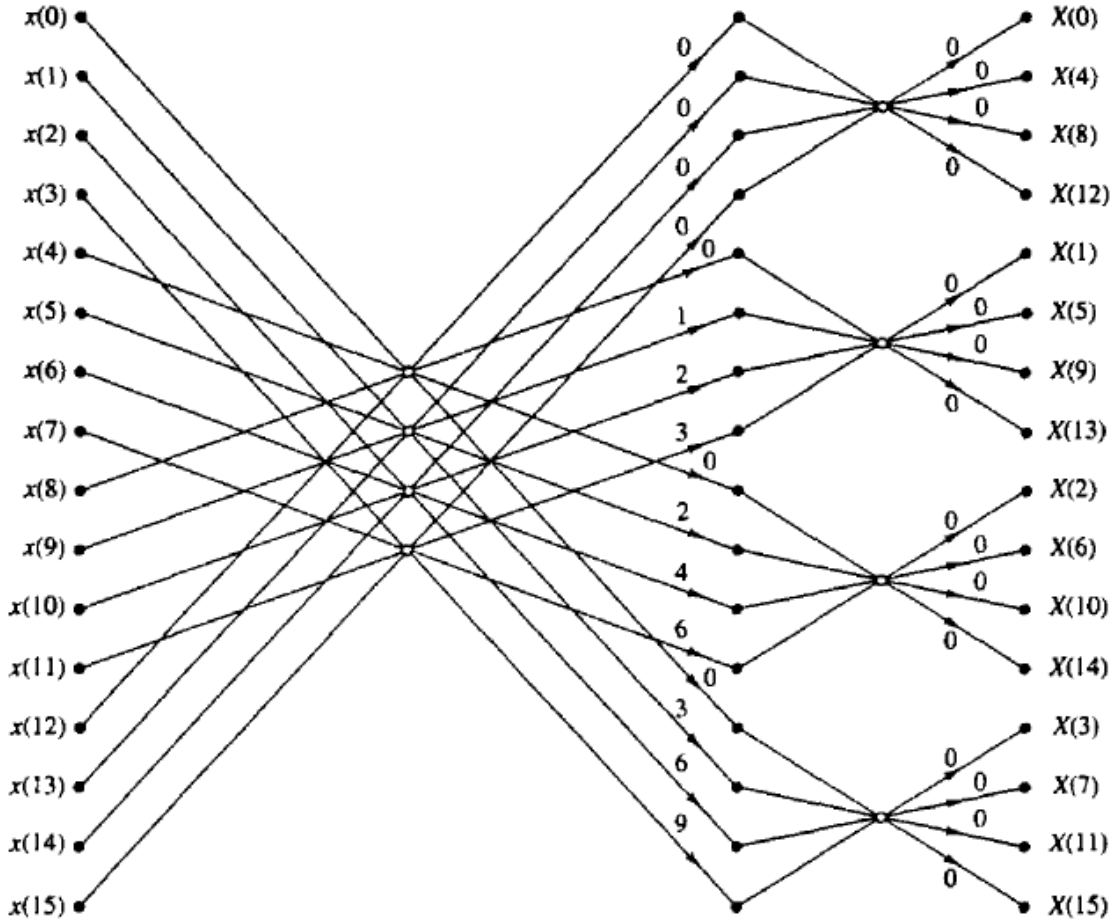
$$x(l, m) = x(4m + l) \tag{6.1.41}$$

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

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

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




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)
x(15)
0
0
0
0
0
1
2
3
0
2
4
6
0
3
6
9
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
X(0)
X(4)
X(8)
X(12)
X(1)
X(5)
X(9)
X(13)
X(2)
X(6)
X(10)
X(14)
X(3)
X(7)
X(11)
X(15)

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

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

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

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

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.

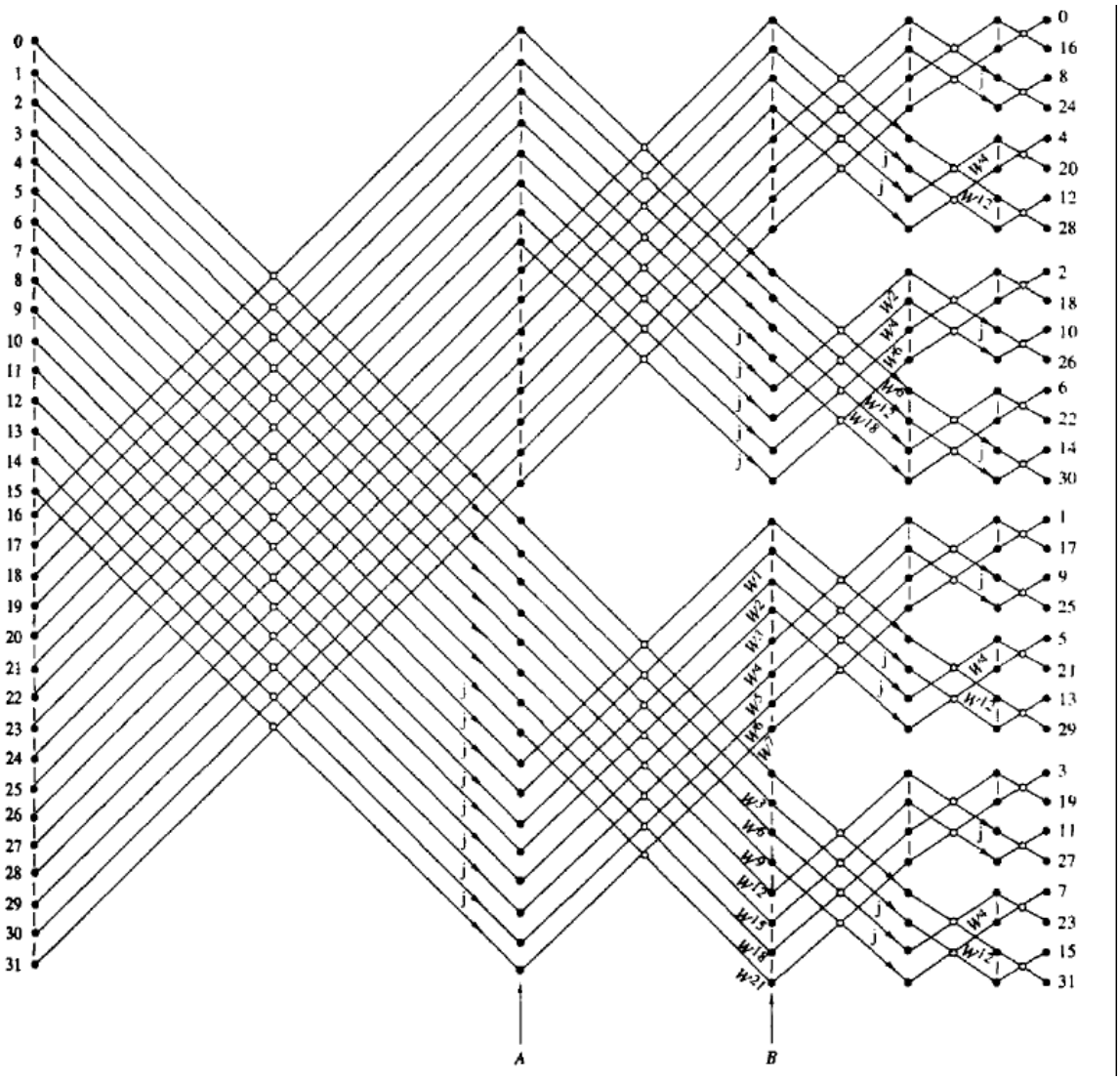
If we use a radix-4 decimation-in-frequency FFT algorithm for the odd-numbered samples of the N-point DFT, we obtain the following $N/4$-point DFTs:

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

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

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



An additional factor of 2 savings in storage of twiddle factors can be obtained by introducing a 90° phase offset at the mid point of each twiddle array , which can be removed if necessary at the ouput of the SRFFT computation. The incorporation of this improvement into the SRFFT results in an other algorithm also due to price called the PFFT algorithm.

Implementation of FFT Algorithms

Now that w e has described the basic radix-2 and radix -4 F FT algorithm s, 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.

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 $N = 2^v$ or $N = 4^v$.

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.

6.2.1 Efficient Computation of the DFT of Two Real Sequences

The FFT algorithm is designed to perform complex multiplications and additions, even though the input data may be real valued. The basic reason for this situation is

Suppose that $x_1(n)$ and $x_2(n)$ are two real-valued sequences of length N, and let $x(n)$ be a complex-valued sequence defined as

$$x(n) = x_1(n) + jx_2(n) \qquad 0 \le n \le N-1 \tag{6.2.1}$$

The DFT operation is linear and hence the DFT of $x(n)$ can be expressed as

$$X(k) = X_1(k) + jX_2(k) \tag{6.2.2}$$

The sequences $x_1(n)$ and $x_2(n)$ can be expressed in terms of $x(n)$ as follows:

$$x_1(n) = \frac{x(n) + x^*(n)}{2} \tag{6.2.3}$$

$$x_2(n) = \frac{x(n) - x^*(n)}{2j} \tag{6.2.4}$$

Hence the DFTs of $x_1(n)$ and $x_2(n)$ are

$$X_1(k) = \frac{1}{2}\{DFT[x(n)] + DFT[x^*(n)]\} \tag{6.2.5}$$

$$X_2(k) = \frac{1}{2j}\{DFT[x(n)] - DFT[x^*(n)]\} \tag{6.2.6}$$

Recall that the DFT of $x^*(n)$ is $X^*(N-k)$. Therefore,

$$X_1(k) = \frac{1}{2}[X(k) + X^*(N-k)] \tag{6.2.7}$$

$$X_2(k) = \frac{1}{j2}[X(k) - X^*(N-k)] \tag{6.2.8}$$

6.2.2 Efficient Computation of the DFT of a $2N$-Point Real Sequence

Suppose that $g(n)$ is a real-valued sequence of $2N$ points. We now demonstrate how to obtain the $2N$-point DFT of $g(n)$ from computation of one N-point DFT involving complex-valued data. First, we define

$$\begin{aligned} x_1(n) &= g(2n) \\ x_2(n) &= g(2n+1) \end{aligned} \tag{6.2.9}$$

Let $x(n)$ be the N-point complex-valued sequence

$$x(n) = x_1(n) + jx_2(n) \tag{6.2.10}$$

From the results of the preceding section, we have

$$\begin{aligned} X_1(k) &= \frac{1}{2}[X(k) + X^*(N-k)] \\ X_2(k) &= \frac{1}{2j}[X(k) - X^*(N-k)] \end{aligned} \tag{6.2.11}$$

Finally, we must express the $2N$-point DFT in terms of the two N-point DFTs, $X_1(k)$ and $X_2(k)$. To accomplish this, we proceed as in the decimation-in-time FFT algorithm, namely,

$$\begin{aligned} G(k) &= \sum_{n=0}^{N-1} g(2n) W_{2N}^{2nk} + \sum_{n=0}^{N-1} g(2n+1) W_{2N}^{(2n+1)k} \\ &= \sum_{n=0}^{N-1} x_1(n) W_N^{nk} + W_{2N}^{k} \sum_{n=0}^{N-1} x_2(n) W_N^{nk} \end{aligned}$$

Consequently,

$$\begin{aligned} G(k) &= X_1(k) + W_2^k N X_2(k) \qquad k = 0, 1, \ldots, N-1 \\ G(k+N) &= X_1(k) - W_2^k N X_2(k) \qquad k = 0, 1, \ldots, N-1 \end{aligned} \tag{6.2.12}$$

Thus we have computed the DFT of a $2N$-point real sequence from one N-point DFT and some additional computation as indicated by (6.2.11) and (6.2.12).

6.2.3 Use of the FFT Algorithm in Linear Filtering and Correlation

An important application of the FFT algorithm is in FIR linear filtering of long data sequences. In Chapter 5 we described two methods, the overlap-add and the overlap-save methods for filtering a long data sequence with an FIR filter, based on the use of the DFT. In this section we consider the use of these two methods in conjunction with the FFT algorithm for computing the DFT and the IDFT.

Let $h(n)$, $0 \le n \le M-1$, be the unit sample response of the FIR filter and let $x(n)$ denote the input data sequence. The block size of the FFT algorithm is N, where $N = L + M - 1$ and L is the number of new data samples being processed by the filter. We assume that for any given value of M, the number L of data samples is selected so that N is a power of 2. For purposes of this discussion, we consider only radix-2 FFT algorithms.

The N-point DFT of $h(n)$, which is padded by $L-1$ zeros, is denoted as $H(k)$. This computation is performed once via the FFT and the resulting N complex numbers are stored. To be specific we assume that the decimation-in-frequency FFT algorithm is used to compute $H(k)$. This yields $H(k)$ in bit-reversed order, which is the way it is stored in memory.

In the overlap-save method, the first $M-1$ data points of each data block are the last $M-1$ data points of the previous data block. Each data block contains L new data points, such that $N = L + M - 1$. The N-point DFT of each data block is performed by the FFT algorithm. If the decimation-in-frequency algorithm is employed, the input data block requires no shuffling and the values of the DFT occur in bit-reversed order. Since this is exactly the order of $H(k)$, we can multiply the DFT of the data, say $X_m(k)$, with $H(k)$ and thus the result

$$Y_m(k) = H(k)X_m(k)$$

is also in bit-reversed order.

The inverse DFT (IDFT) can be computed by use of an FFT algorithm that takes the input in bit-reversed order and produces an output in normal order. Thus there is no need to shuffle any block of data either in computing the DFT or the IDFT.

If the overlap-add method is used to perform the linear filtering, the computational method using the FFT algorithm is basically the same. The only difference is that the N-point data blocks consist of L new data points and $M-1$ additional zeros. After the IDFT is computed for each data block, the N-point filtered blocks are overlapped as indicated in Section 5.3.2, and the $M-1$ overlapping data points between successive output records are added together.

6.3.1 The Goertzel Algorithm

The Goertzel algorithm exploits the periodicity of the phase factors $\{W_N^k\}$ and allows us to express the computation of the DFT as a linear filtering operation. Since $W_N^{-kN} = 1$, we can multiply the DFT by this factor. Thus

$$X(k) = W_N^{-kN} \sum_{m=0}^{N-1} x(m) W_N^{km} = \sum_{m=0}^{N-1} x(m) W_N^{-k(N-m)} \tag{6.3.1}$$

We note that (6.3.1) is in the form of a convolution. Indeed, if we define the sequence $y_k(n)$ as

$$y_k(n) = \sum_{m=0}^{N-1} x(m) W_N^{-k(n-m)} \tag{6.3.2}$$

then it is clear that $y_k(n)$ is the convolution of the finite-duration input sequence $x(n)$ of length N with a filter that has an impulse response

$$h_k(n) = W_N^{-kn} u(n) \tag{6.3.3}$$

The output of this filter at $n = N$ yields the value of the DFT at the frequency $\omega_k = 2\pi k/N$. That is,

$$X(k) = y_k(n)|_{n=N} \tag{6.3.4}$$

as can be verified by comparing (6.3.1) with (6.3.2).

The filter with impulse response $h_k(n)$ has the system function

$$H_k(z) = \frac{1}{1 - W_N^{-k} z^{-1}} \tag{6.3.5}$$

This filter has a pole on the unit circle at the frequency $\omega_k = 2\pi k/N$. Thus, the entire DFT can be computed by passing the block of input data into a parallel bank of N single-pole filters (resonators), where each filter has a pole at the corresponding frequency of the DFT.

Instead of performing the computation of the DFT as in (6.3.2), via convolution, we can use the difference equation corresponding to the filter given by (6.3.5) to compute $y_k(n)$ recursively. Thus we have

$$y_k(n) = W_N^{-k} y_k(n-1) + x(n) \qquad y_k(-1) = 0 \tag{6.3.6}$$

The desired output is $X(k) = y_k(N)$, for $k = 0, 1, \ldots, N-1$. To perform this computation, we can compute once and store the phase factors W_N^{-k}.

The complex multiplications and additions inherent in (6.3.6) can be avoided by combining the pairs of resonators possessing complex-conjugate poles. This leads to two-pole filters with system functions of the form

$$H_k(z) = \frac{1 - W_N^k z^{-1}}{1 - 2\cos(2\pi k/N) z^{-1} + z^{-2}} \tag{6.3.7}$$

6.3.2 The Chirp-z Transform Algorithm

The DFT of an N-point data sequence x(n) has been viewed as the z-transform of $x(n)$ evaluated at N equally spaced points on the unit circle. It has also been viewed as N equally spaced samples of the Fourier transform of the data sequence $x(n)$. In this section we consider the evaluation of $X(z)$ on other contours in the z-plane, including the unit circle.

Suppose that we wish to compute the values of the z-transform of $x(n)$ at a set of points $\{z_k\}$. Then,

$$X(z_k) = \sum_{n=0}^{N-1} x(n) z_k^{-n} \qquad k = 0, 1, \ldots, L-1 \tag{6.3.10}$$

For example, if the contour is a circle of radius r and the z_k are N equally spaced points, then

$$\begin{aligned} z_k &= re^{j2\pi kn/N} \qquad k = 0, 1, 2, \ldots, N-1 \\ X(z_k) &= \sum_{n=0}^{N-1} [x(n) r^{-n}] e^{-j2\pi kn/N} \qquad k = 0, 1, 2, \ldots, N-1 \end{aligned} \tag{6.3.11}$$

In this case the FFT algorithm can be applied on the modified sequence $x(n)r^{-n}$.

More generally, suppose that the points z_k in the z-plane fall on an arc which begins at some point

$$z_0 = r_0 e^{j\theta_0}$$

and spirals either in toward the origin or out away from the origin such that the points $\{z_k\}$ are defined as

$$z_k = r_0 e^{j\theta_0} (R_0 e^{j\phi_0})^k \qquad k = 0, 1, \ldots, L-1 \tag{6.3.12}$$

When points $\{z_k\}$ in (6.3.12) are substituted into the expression for the z-transform, we obtain

$$\begin{aligned} X(z_k) &= \sum_{n=0}^{N-1} x(n) z_k^{-n} \\ &= \sum_{n=0}^{N-1} x(n) (r_0 e^{j\theta_0})^{-n} V^{-nk} \end{aligned} \tag{6.3.13}$$

where, by definition,

$$V = R_0 e^{j\Phi_0} \tag{6.3.14}$$

We can express (6.3.13) in the form of a convolution, by noting that

$$nk = \tfrac{1}{2}[n^2 + k^2 - (k-n)^2] \tag{6.3.15}$$

Substitution of (6.3.15) into (6.3.13) yields

$$X(z_k) = V^{-k^2/2} \sum_{n=0}^{N-1} [x(n)(r_0 e^{j\theta_0})^{-n} V^{-n^2/2}] V^{(k-n)^2/2} \tag{6.3.16}$$

Let us define a new sequence $g(n)$ as

$$g(n) = x(n)(r_0 e^{j\theta_0})^{-n} V^{-n^2/2} \tag{6.3.17}$$

Then (6.3.16) can be expressed as

$$X(z_k) = V^{-k^2/2} \sum_{n=0}^{N-1} g(n) V^{(k-n)^2/2} \tag{6.3.18}$$

The summation in (6.3.18) can be interpreted as the convolution of the sequence $g(n)$ with the impulse response $h(n)$ of a filter, where

$$h(n) = V^{n^2/2} \tag{6.3.19}$$

Consequently, (6.3.18) may be expressed as

$$\begin{aligned} X(z_k) &= V^{-k^2/2} y(k) \\ &= \frac{y(k)}{h(k)} \qquad k = 0, 1, \ldots, L-1 \end{aligned} \tag{6.3.20}$$

where $y(k)$ is the output of the filter

$$y(k) = \sum_{n=0}^{N-1} g(n) h(k-n) \qquad k = 0, 1, \ldots, L-1 \tag{6.3.21}$$

QUANTIZATION EFFECTS IN THE COMPUTATION OF THE DFT*

As we have observed in our previous discussions, the DFT plays an important role in many digital signal processing applications, including FIR filtering, the computation of the correlation between signals, and spectral analysis. For this reason it is important for us to know the effect of quantization errors in its computation. In particular, we shall consider the effect of round-off errors due to the multiplications performed in the DFT with fixed-point arithmetic.

6.4.1 Quantization Errors in the Direct Computation of the DFT

Given a finite-duration sequence $\{x(n)\}$, $0 \leq n \leq N-1$, the DFT of $\{x(n)\}$ is defined as

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \qquad k = 0, 1, \ldots, N-1 \tag{6.4.1}$$

where $W_N = e^{-j2\pi/N}$. We assume that in general, $\{x(n)\}$ is a complex-valued sequence. We also assume that the real and imaginary components of $\{x(n)\}$ and $\{W_N^{kn}\}$ are represented by b bits. Consequently, the computation of the product $x(n)W_N^{kn}$ requires four real multiplications. Each real multiplication is rounded from $2b$ bits to b bits, and hence there are four quantization errors for each complex-valued multiplication.

In the direct computation of the DFT, there are N complex-valued multiplications for each point in the DFT. Therefore, the total number of real multiplications in the computation of a single point in the DFT is $4N$. Consequently, there are $4N$ quantization errors.

Let us evaluate the variance of the quantization errors in a fixed-point computation of the DFT. First, we make the following assumptions about the statistical properties of the quantization errors.

1. The quantization errors due to rounding are uniformly distributed random variables in the range $(-\Delta/2, \Delta/2)$ where $\Delta = 2^{-b}$.
2. The $4N$ quantization errors are mutually uncorrelated.
3. The $4N$ quantization errors are uncorrelated with the sequence $\{x(n)\}$.

Since each of the quantization errors has a variance

$$\sigma_e^2 = \frac{\Delta^2}{12} = \frac{2^{-2b}}{12} \tag{6.4.2}$$

the variance of the quantization errors from the $4N$ multiplications is

$$\begin{aligned} \sigma_q^2 &= 4N\sigma_e^2 \\ &= \frac{N}{3} \cdot 2^{-2b} \end{aligned} \tag{6.4.3}$$

as

$$\sigma_q^2 = \frac{2^{-2(b-v/2)}}{3} \tag{6.4.4}$$

This expression implies that every fourfold increase in the size N of the DFT requires an additional bit in computational precision to offset the additional quantization errors.

To prevent overflow, the input sequence to the DFT requires scaling. Clearly, an upper bound on $|X(k)|$ is

$$|X(k)| \leq \sum_{n=0}^{N-1} |x(n)| \tag{6.4.5}$$

If the dynamic range in addition is $(-1, 1)$, then $|X(k)| < 1$ requires that

$$\sum_{n=0}^{N-1} |x(n)| < 1 \tag{6.4.6}$$

If $|x(n)|$ is initially scaled such that $|x(n)| < 1$ for all n, then each point in the sequence can be divided by N to ensure that (6.4.6) is satisfied.

The scaling implied by (6.4.6) is extremely severe. For example, suppose that the signal sequence $\{x(n)\}$ is white and, after scaling, each value $|x(n)|$ of the sequence is uniformly distributed in the range $(-1/N, 1/N)$. Then the variance of the signal sequence is

$$\sigma_x^2 = \frac{(2/N)^2}{12} = \frac{1}{3N^2} \tag{6.4.7}$$

and the variance of the output DFT coefficients $|X(k)|$ is

$$\begin{aligned} \sigma_X^2 &= N\sigma_x^2 \\ &= \frac{1}{3N} \end{aligned} \tag{6.4.8}$$

Thus the signal-to-noise power ratio is

$$\frac{\sigma_X^2}{\sigma_q^2} = \frac{2^{2b}}{N^2} \tag{6.4.9}$$

We observe that the scaling is responsible for reducing the SNR by N and the combination of scaling and quantization errors result in a total reduction that is proportional to N^2. Hence scaling the input sequence $\{x(n)\}$ to satisfy (6.4.6) imposes a severe penalty on the signal-to-noise ratio in the DFT.

****