Chapter 7

The Fast Fourier Transform

7.1 INTRODUCTION

In Chap. 6 we saw that the discrete Fourier transform (DFT) could be used to perform convolutions. In this chapter we look at the computational requirements of the DFT and derive some *fast algorithms* for computing the DFT. These algorithms are known, generically, as *fast Fourier transforms* (FFTs). We begin with the radix-2 decimation-in-time FFT, an algorithm published in 1965 by Cooley and Tukey. We then look at mixed-radix FFT algorithms and the prime factor FFT.

7.2 RADIX-2 FFT ALGORITHMS

The N-point DFT of an N-point sequence $x(n)$ is

$$X(k)=\sum_{n=0}^{N-1}x(n)W_N^{nk} \tag{7.1}$$

Because $x(n)$ may be either real or complex, evaluating $X(k)$ requires on the order of N complex multiplications and N complex additions for each value of k. Therefore, because there are N values of $X(k)$, computing an N-point DFT requires N^2 complex multiplications and additions.

The basic strategy that is used in the FFT algorithm is one of "divide and conquer," which involves decomposing an N-point DFT into successively smaller DFTs. To see how this works, suppose that the length of $x(n)$ is even (i.e., N is divisible by 2). If $x(n)$ is *decimated* into two sequences of length $N/2$, computing the $N/2$-point DFT of each of these sequences requires approximately $(N/2)^2$ multiplications and the same number of additions. Thus, the two DFTs require $2(N/2)^2=\frac{1}{2}N^2$ multiplies and adds. Therefore, if it is possible to find the N-point DFT of $x(n)$ from these two $N/2$-point DFTs in fewer than $N^2/2$ operations, a savings has been realized.

7.2.1 *Decimation-in-Time FFT*

The decimation-in-time FFT algorithm is based on splitting (decimating) $x(n)$ into smaller sequences and finding $X(k)$ from the DFTs of these decimated sequences. This section describes how this decimation leads to an efficient algorithm when the sequence length is a power of 2.

Let $x(n)$ be a sequence of length $N=2^\nu$, and suppose that $x(n)$ is split (decimated) into two subsequences, each of length $N/2$. As illustrated in Fig. 7-1, the first sequence, $g(n)$, is formed from the even-index terms,

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

and the second, $h(n)$, is formed from the odd-index terms,

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

In terms of these sequences, the N-point DFT of $x(n)$ is

$$\begin{aligned}X(k)=\sum_{n=0}^{N-1}x(n)W_N^{nk}&=\sum_{n\text{ even}}x(n)W_N^{nk}+\sum_{n\text{ odd}}x(n)W_N^{nk}\\&=\sum_{l=0}^{\frac{N}{2}-1}g(l)W_N^{2lk}+\sum_{l=0}^{\frac{N}{2}-1}h(l)W_N^{(2l+1)k}\end{aligned} \tag{7.2}$$

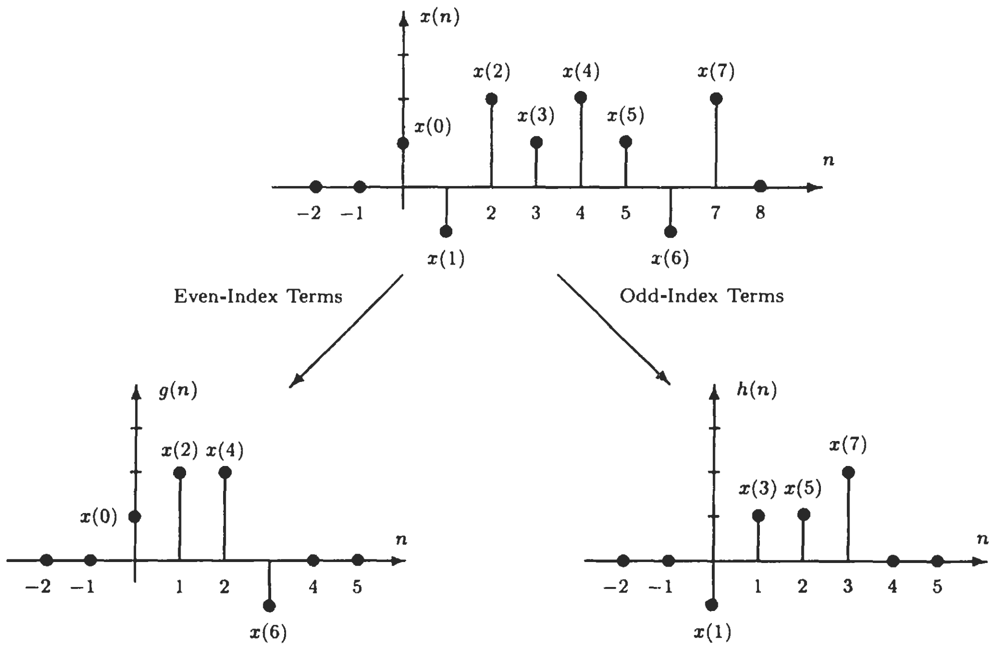


Fig. 7-1. Decimating a sequence of length $N = 8$ by a factor of 2.

Because $W_N^{2lk} = W_{N/2}^{lk}$, Eq. (*7.2*) may be written as

$$X(k) = \sum_{l=0}^{\frac{N}{2}-1} g(l)W_{N/2}^{lk} + W_N^k \sum_{l=0}^{\frac{N}{2}-1} h(l)W_{N/2}^{lk}$$

Note that the first term is the $N/2$-point DFT of $g(n)$, and the second is the $N/2$-point DFT of $h(n)$:

$$X(k) = G(k) + W_N^k H(k) \qquad k = 0, 1, \dots, N-1 \tag{7.3}$$

Although the $N/2$-point DFTs of $g(n)$ and $h(n)$ are sequences of length $N/2$, the periodicity of the complex exponentials allows us to write

$$G(k) = G\left(k + \frac{N}{2}\right) \qquad H(k) = H\left(k + \frac{N}{2}\right)$$

Therefore, $X(k)$ may be computed from the $N/2$-point DFTs $G(k)$ and $H(k)$. Note that because

$$W_N^{k+N/2} = W_N^k W_N^{N/2} = -W_N^k$$

then

$$W_N^{k+\frac{N}{2}} H\left(k + \tfrac{N}{2}\right) = -W_N^k H(k)$$

and it is only necessary to form the products $W_N^k H(k)$ for $k = 0, 1, \dots, N/2 - 1$. The complex exponentials multiplying $H(k)$ in Eq. (*7.3*) are called *twiddle factors*. A block diagram showing the computations that are necessary for the first *stage* of an eight-point decimation-in-time FFT is shown in Fig. 7-2.

If $N/2$ is even, $g(n)$ and $h(n)$ may again be decimated. For example, $G(k)$ may be evaluated as follows:

$$G(k) = \sum_{n=0}^{\frac{N}{2}-1} g(n)W_{N/2}^{nk} = \sum_{n \text{ even}}^{\frac{N}{2}-1} g(n)W_{N/2}^{nk} + \sum_{n \text{ odd}}^{\frac{N}{2}-1} g(n)W_{N/2}^{nk}$$

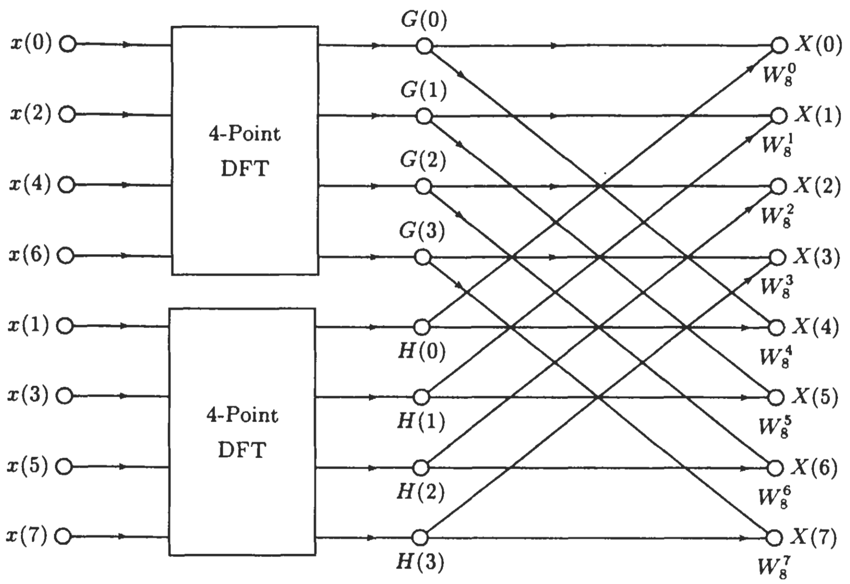


Fig. 7-2. An eight-point decimation-in-time FFT algorithm after the first decimation.

As before, this leads to

$$G(k) = \sum_{n=0}^{\frac{N}{4}-1} g(2n)W_{N/4}^{nk} + W_{N/2}^{k} \sum_{n=0}^{\frac{N}{4}-1} g(2n+1)W_{N/4}^{nk}$$

where the first term is the $N/4$-point DFT of the even samples of $g(n)$, and the second is the $N/4$-point DFT of the odd samples. A block diagram illustrating this decomposition is shown in Fig. 7-3. If N is a power of 2, the decimation may be continued until there are only two-point DFTs of the form shown in Fig. 7-4.

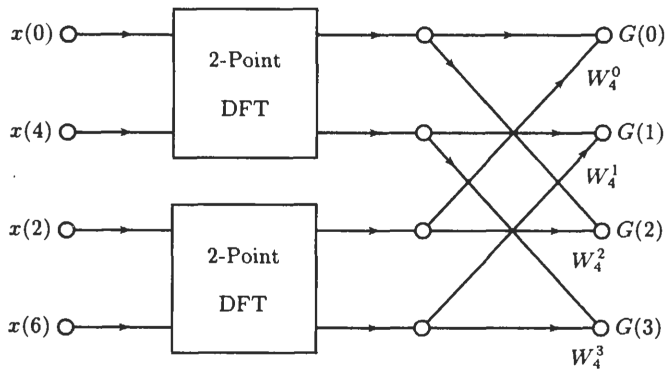


Fig. 7-3. Decimation of the four-point DFT into two two-point DFTs in the decimation-in-time FFT.

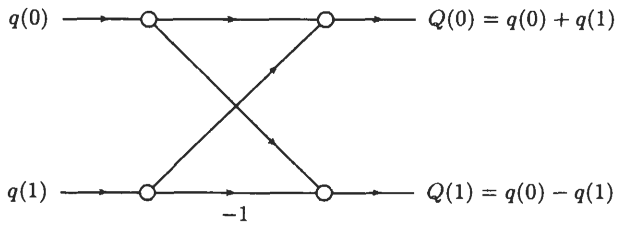


Fig. 7-4. A two-point DFT.

The basic computational unit of the FFT, shown in Fig. 7-5(*a*), is called a *butterfly*. This structure may be simplified by factoring out a term W_N^r from the lower branch as illustrated in Fig. 7-5(*b*). The factor that remains is $W_N^{N/2} = -1$. A complete eight-point radix-2 decimation-in-time FFT is shown in Fig. 7-6.

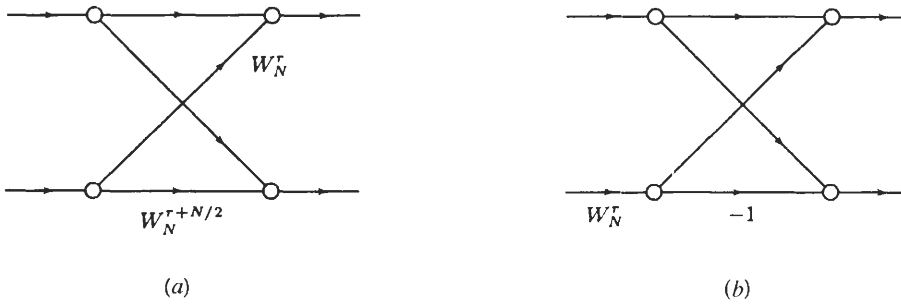


Fig. 7-5. (*a*) The butterfly, which is the basic computational element of the FFT algorithm. (*b*) A simplified butterfly, with only one complex multiplication.

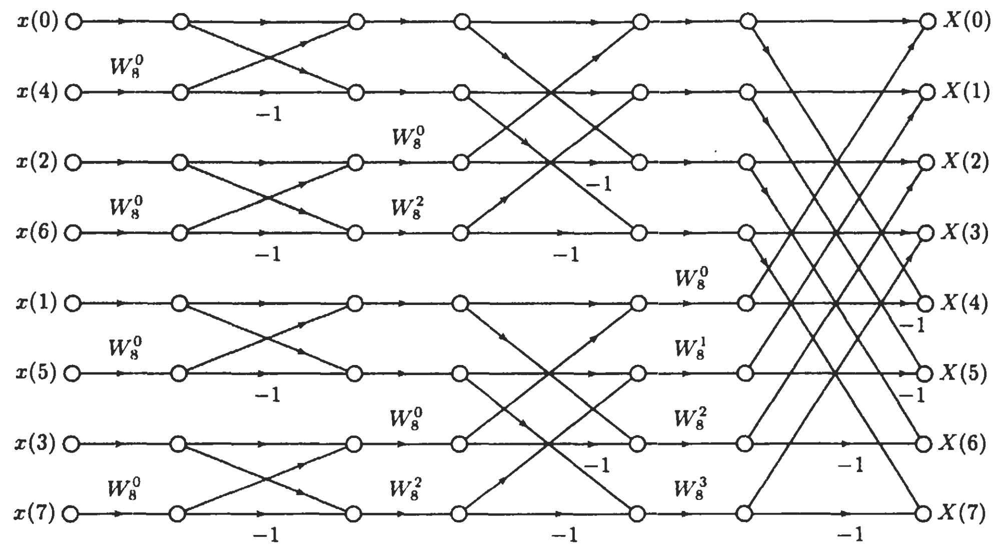


Fig. 7-6. A complete eight-point radix-2 decimation-in-time FFT.

Computing an N-point DFT using a radix-2 decimation-in-time FFT is much more efficient than calculating the DFT directly. For example, if $N = 2^\nu$, there are $\log_2 N = \nu$ *stages* of computation. Because each stage requires $N/2$ complex multiplies by the twiddle factors W_N^r and N complex additions, there are a total of $\frac{1}{2}N \log_2 N$ complex multiplications[1] and $N \log_2 N$ complex additions.

From the structure of the decimation-in-time FFT algorithm, note that once a butterfly operation has been performed on a pair of complex numbers, there is no need to save the input pair. Therefore, the output pair may be stored in the same registers as the input. Thus, only one array of size N is required, and it is said that the computations may be performed *in place*. To perform the computations in place, however, the input sequence $x(n)$ must be stored (or accessed) in nonsequential order as seen in Fig. 7-6. The *shuffling* of the input sequence that takes place is due to the successive decimations of $x(n)$. The ordering that results corresponds to a bit-reversed indexing of the original sequence. In other words, if the index n is written in binary form, the order in which in the input sequence must be accessed is found by reading the binary representation for n in reverse order as illustrated in the table below for $N = 8$:

[1]The number of multiplications is actually a bit less than this because some of the twiddle factors are equal to 1.

n	Binary	Bit-Reversed Binary	n'
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Alternate forms of FFT algorithms may be derived from the decimation-in-time FFT by manipulating the flowgraph and rearranging the order in which the results of each stage of the computation are stored. For example, the nodes of the flowgraph may be rearranged so that the input sequence $x(n)$ is in normal order. What is lost with this reordering, however, is the ability to perform the computations in place.

7.2.2 *Decimation-in-Frequency FFT*

Another class of FFT algorithms may be derived by decimating the output sequence $X(k)$ into smaller and smaller subsequences. These algorithms are called *decimation-in-frequency* FFTs and may be derived as follows. Let N be a power of 2, $N = 2^\nu$, and consider separately evaluating the even-index and odd-index samples of $X(k)$. The even samples are

$$X(2k) = \sum_{n=0}^{N-1} x(n) W_N^{2nk}$$

Separating this sum into the first $N/2$ points and the last $N/2$ points, and using the fact that $W_N^{2nk} = W_{N/2}^{nk}$, this becomes

$$X(2k) = \sum_{n=0}^{\frac{N}{2}-1} x(n) W_{N/2}^{nk} + \sum_{n=N/2}^{N-1} x(n) W_{N/2}^{nk}$$

With a change in the indexing on the second sum we have

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

Finally, because $W_{N/2}^{(n+\frac{N}{2})k} = W_{N/2}^{nk}$,

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

which is the $N/2$-point DFT of the sequence that is formed by adding the first $N/2$ points of $x(n)$ to the last $N/2$.

Proceeding in the same way for the odd samples of $X(k)$ leads to

$$X(2k+1) = \sum_{n=0}^{\frac{N}{2}-1} W_N^n \left[x(n) - x\left(n + \frac{N}{2}\right)\right] W_{N/2}^{nk} \tag{7.4}$$

A flowgraph illustrating this first stage of decimation is shown in Fig. 7-7. As with the decimation-in-time FFT, the decimation may be continued until only two-point DFTs remain. A complete eight-point decimation-in-frequency FFT is shown in Fig. 7-8. The complexity of the decimation-in-frequency FFT is the same as the decimation-in-time, and the computations may be performed in place. Finally, note that although the input sequence $x(n)$ is in normal order, the frequency samples $X(k)$ are in bit-reversed order.

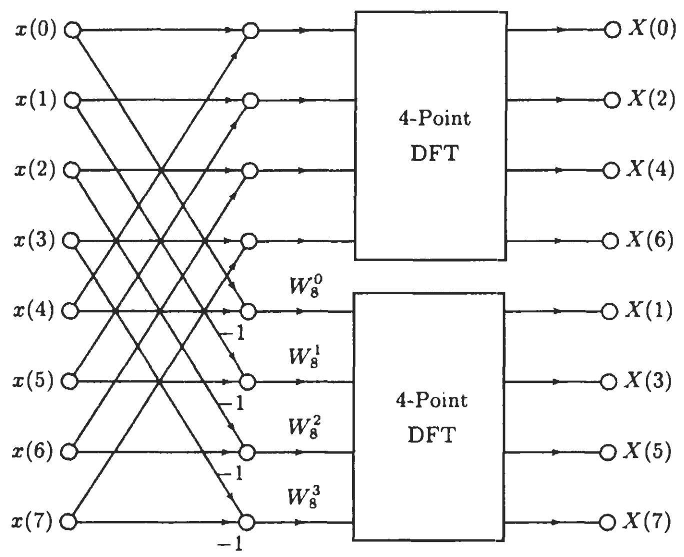


Fig. 7-7. An eight-point decimation-in-frequency FFT algorithm after the first stage of decimation.

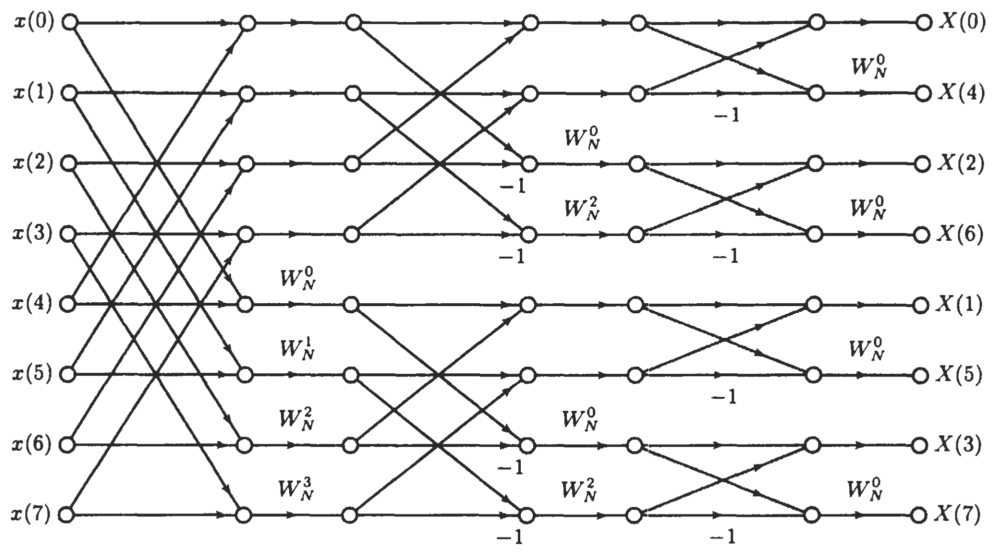


Fig. 7-8. Eight-point radix-2 decimation-in-frequency FFT.

7.3 FFT ALGORITHMS FOR COMPOSITE N

It is not always possible to work with sequences whose length is a power of 2. However, efficient computation of the DFT is still possible if the sequence length may be written as a product of factors. For example, suppose that N may be factored as follows:

$$N = N_1 \cdot N_2$$

We then decompose $x(n)$ into N_2 sequences of length N_1 and arrange these sequences in an array as follows:

$$\mathbf{x} = \begin{bmatrix} x(0) & x(N_2) & \cdots & x(N_2(N_1-1)) \\ x(1) & x(N_2+1) & \cdots & x(N_2(N_1-1)+1) \\ \vdots & \vdots & & \vdots \\ x(N_2-1) & x(2N_2-1) & \cdots & x(N_1N_2-1) \end{bmatrix} \tag{7.5}$$

EXAMPLE 7.3.1 For a sequence of length $N = 15$, with $N_1 = 3$ and $N_2 = 5$, the sequence $x(n)$ may be decimated into five sequences of length three, and these sequences may then be arranged in a two-dimensional array as follows:

$$\mathbf{x} = \begin{bmatrix} x(0) & x(5) & x(10) \\ x(1) & x(6) & x(11) \\ x(2) & x(7) & x(12) \\ x(3) & x(8) & x(13) \\ x(4) & x(9) & x(14) \end{bmatrix}$$

Alternatively, if we let $N_1 = 5$ and $N_2 = 3$, $x(n)$ may be decimated into three sequences of length five and arranged in a two-dimensional array of three rows and five columns,

$$\mathbf{x} = \begin{bmatrix} x(0) & x(3) & x(6) & x(9) & x(12) \\ x(1) & x(4) & x(7) & x(10) & x(13) \\ x(2) & x(5) & x(8) & x(11) & x(14) \end{bmatrix}$$

By defining *index maps* for n and k as follows,

$$n = N_2 \cdot n_1 + n_2 \qquad \begin{cases} 0 \le n_1 \le N_1 - 1 \\ 0 \le n_2 \le N_2 - 1 \end{cases}$$

$$k = k_1 + N_1 \cdot k_2 \qquad \begin{cases} 0 \le k_1 \le N_1 - 1 \\ 0 \le k_2 \le N_2 - 1 \end{cases}$$

the N-point DFT may be expressed as

$$\begin{aligned} X(k) = X(k_1 + N_1 k_2) &= \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x(N_2 n_1 + n_2) W_N^{(k_1+N_1k_2)(N_2n_1+n_2)} \qquad (7.6) \\ &= \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x(N_2 n_1 + n_2) W_N^{N_2k_1n_1} W_N^{k_1n_2} W_N^{N_1k_2n_2} W_N^{N_1N_2k_2n_1} \end{aligned}$$

Because $W_N^{N_2k_1n_1} = W_{N_1}^{k_1n_1}$, $W_N^{N_1k_2n_2} = W_{N_2}^{k_2n_2}$, and $W_N^{N_1N_2k_2n_1} = 1$, the DFT becomes

$$X(k) = \sum_{n_2=0}^{N_2-1} \left\{ \left[\sum_{n_1=0}^{N_1-1} x(N_2 n_1 + n_2) W_{N_1}^{n_1k_1} \right] W_N^{k_1n_2} \right\} W_{N_2}^{k_2n_2} \qquad (7.7)$$

Note that the inner summation,

$$G(n_2, k_1) = \sum_{n_1=0}^{N_1-1} x(N_2 n_1 + n_2) W_{N_1}^{n_1k_1}$$

is the N_1-point DFT of the sequence $x(N_2n_1 + n_2)$, which is row n_2 of the two-dimensional array in Eq. (*7.5*). Computing the N_1-point DFT of each row of the array produces another array,

$$\mathbf{G} = \begin{bmatrix} G(0,0) & G(0,1) & \cdots & G(0, N_1 - 1) \\ G(1,0) & G(1,1) & \cdots & G(1, N_1 - 1) \\ \vdots & \vdots & & \vdots \\ G(N_2 - 1, 0) & G(N_2 - 1, 1) & \cdots & G(N_2 - 1, N_1 - 1) \end{bmatrix}$$

consisting of the complex numbers $G(n_2, k_1)$. Note that because the data in row n_2 is not needed after the N_1-point DFT of $x(N_2n_1 + n_2)$ is computed, $G(n_2, k_1)$ may be stored in the same row (i.e., the computations may be performed in place).

The next step in the evaluation of $X(k)$ in Eq. (7.7) is to multiply by the twiddle factors $W_N^{k_1 n_2}$:

$$\tilde{G}(n_2, k_1) = W_N^{k_1 n_2} G(n_2, k_1)$$

The final step is to compute the N_2-point DFT of the columns of the array $\tilde{G}(n_2, k_1)$:

$$X(k_1 + N_1 k_2) = \sum_{n_2=0}^{N_2-1} \tilde{G}(n_2, k_1) W_{N_2}^{k_2 n_2}$$

The DFT coefficients are then read out *row-wise* from the two-dimensional array:

$$X(k) = X(k_1 + N_1 k_2)$$

A pictorial representation of this decomposition is shown in Fig. 7-9 for $N = 15$.

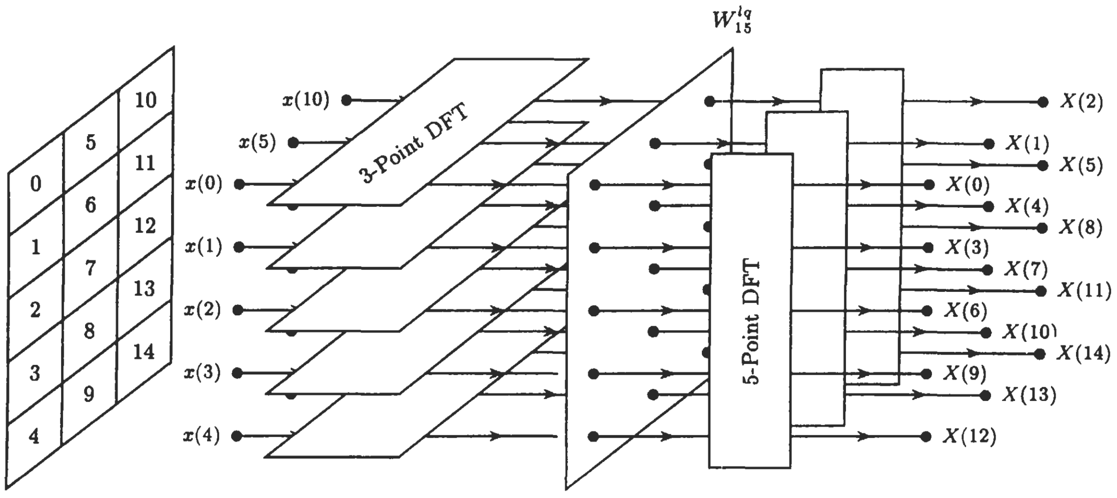


Fig. 7-9. Computation of a 15-point DFT with $N_1 = 3$ and $N_2 = 5$ using 3-point and 5-point DFTs.

EXAMPLE 7.3.2 Suppose that we want to compute the 12-point DFT of $x(n)$. With $N_1 = 3$ and $N_2 = 4$, the first step is to form a two-dimensional array consisting of $N_1 = 3$ columns and $N_2 = 4$ rows,

n_2 \ n_1	0	1	2
0	$x(0)$	$x(4)$	$x(8)$
1	$x(1)$	$x(5)$	$x(9)$
2	$x(2)$	$x(6)$	$x(10)$
3	$x(3)$	$x(7)$	$x(11)$

and compute the DFT of each row,

n_2 \ k_1	0	1	2
0	$G(0, 0)$	$G(0, 1)$	$G(0, 2)$
1	$G(1, 0)$	$G(1, 1)$	$G(1, 2)$
2	$G(2, 0)$	$G(2, 1)$	$G(2, 2)$
3	$G(3, 0)$	$G(3, 1)$	$G(3, 2)$

For example, the DFT of the first row is

$$G(0, k) = x(0) + x(4)W_3^k + x(8)W_3^{2k} \qquad k = 0, 1, 2$$

The next step is to multiply each term by the appropriate twiddle factor. The array of factors is

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & W_{12} & W_{12}^2 \\ 1 & W_{12}^2 & W_{12}^4 \\ 1 & W_{12}^3 & W_{12}^6 \end{bmatrix}$$

This produces the array $\tilde{G}(n_2, k_1)$:

$n_2 \backslash k_1$	0	1	2
0	$\tilde{G}(0,0)$	$\tilde{G}(0,1)$	$\tilde{G}(0,2)$
1	$\tilde{G}(1,0)$	$\tilde{G}(1,1)$	$\tilde{G}(1,2)$
2	$\tilde{G}(2,0)$	$\tilde{G}(2,1)$	$\tilde{G}(2,2)$
3	$\tilde{G}(3,0)$	$\tilde{G}(3,1)$	$\tilde{G}(3,2)$

The final step is to compute the DFT of each column:

$k_2 \backslash k_1$	0	1	2
0	$X(0)$	$X(1)$	$X(2)$
1	$X(3)$	$X(4)$	$X(5)$
2	$X(6)$	$X(7)$	$X(8)$
3	$X(9)$	$X(10)$	$X(11)$

This results in the flowgraph shown in Fig. 7-10. Note that because N_2 can be factored, $N_2 = 2 \times 2$, the four-point DFTs of the columns of $\tilde{G}(n_2, k_1)$ may be evaluated using two-point DFTs. For example, if the first column is arranged in a two-dimensional array,

$$\begin{bmatrix} \tilde{G}(0,0) & \tilde{G}(2,0) \\ \tilde{G}(1,0) & \tilde{G}(3,0) \end{bmatrix}$$

after taking the two-point DFTs of the rows, the terms are multiplied by the twiddle factors

$$\begin{bmatrix} 1 & 1 \\ 1 & W_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -j \end{bmatrix}$$

and then the two-point DFTs of the columns are computed.

Up to this point, we have only assumed that N could be factored as $N = N_1 \cdot N_2$. It is possible, however, that either or both of these factors could be factored further. What is important for the FFT algorithm to be efficient is that N be a highly composite number:

$$N = N_1 \cdot N_2 \cdots N_\nu$$

In this case, it is possible to define multidimensional index maps for n and k as follows,

$$\begin{aligned} n &= N_\nu n_1 + N_{\nu-1} n_2 + \cdots + n_\nu \\ k &= k_1 + N_1 k_2 + \cdots + N_\nu k_\nu \end{aligned}$$

and the development of the FFT algorithm proceeds as described above. If $N = R^\nu$, the corresponding FFT algorithm is called a *Radix-R algorithm*. If the factors are not equal, the FFT is called a *mixed-radix algorithm.*

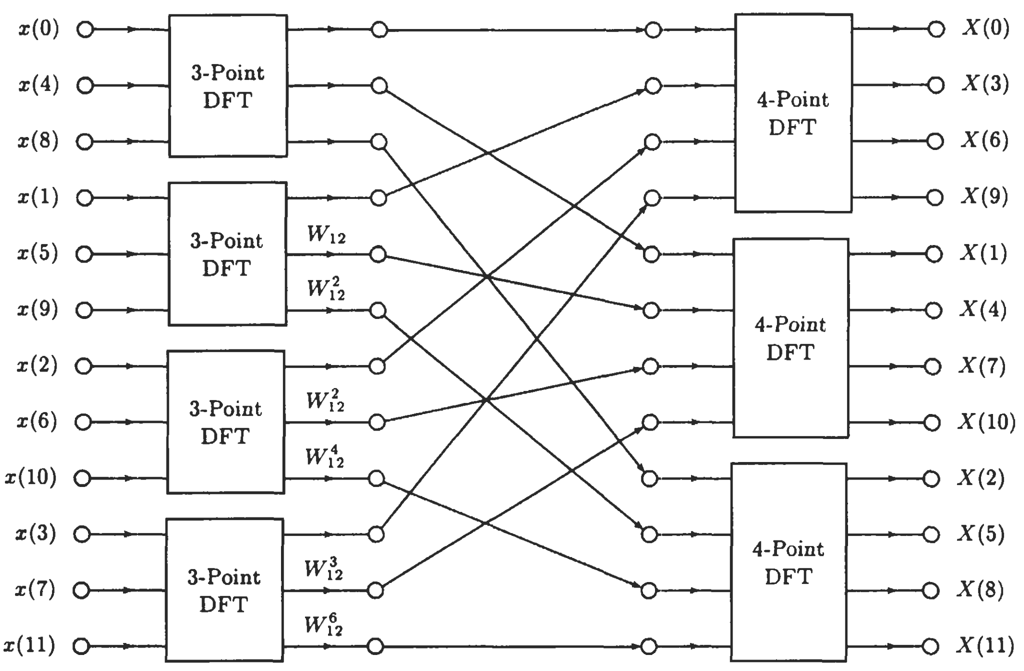


Fig. 7-10. FFT algorithm for $N = 12$.

7.4 PRIME FACTOR FFT

For some values of N, with the appropriate index mapping, it is possible to completely eliminate the twiddle factors. These mapping have the form

$$n = ((An_1 + Bn_2))_N \qquad \begin{cases} 0 \le n_1 \le N_1 - 1 \\ 0 \le n_2 \le N_2 - 1 \end{cases}$$

$$k = ((Ck_1 + Dk_2))_N \qquad \begin{cases} 0 \le k_1 \le N_1 - 1 \\ 0 \le k_2 \le N_2 - 1 \end{cases}$$

where A, B, C, and D are integers, and $((\cdot))_N$ denotes the evaluation of the index modulo N. If $N = N_1 \cdot N_2$, and if N_1 and N_2 are *relatively prime* (i.e., they have no common factors), the twiddle factors may be eliminated with the appropriate values for A, B, C, and D. The requirements on these numbers are as follows:

1. All numbers between 0 and $N - 1$ for n and k must appear uniquely as n_1 and n_2 are varied and as k_1 and k_2 are varied.
2. The numbers A, B, C, and D are such that

$$W_N^{(An_1+Bn_2)(Ck_1+Dk_2)} = W_{N_1}^{n_1k_1} W_{N_2}^{n_2k_2}$$

The second condition requires that

$$((AC))_N = N_2 \qquad ((BD))_N = N_1 \qquad ((AD))_N = ((BC))_N = 0$$

Finding a set of numbers that satisfies these two conditions falls in the domain of *number theory*, which will not be considered here. However, one set of numbers that satisfies these conditions is

$$A = N_2 \qquad B = N_1$$
$$C = N_2\big((N_2^{-1})\big)_{N_1} \qquad D = N_1\big((N_1^{-1})\big)_{N_2}$$

where $((N_1^{-1}))_{N_2}$ denotes the *multiplicative inverse* of N_1 modulo N_2. For example, if $N = 12$ with $N_1 = 3$ and $N_2 = 4$, $((4^{-1}))_3 = 1$ because $((4 \cdot 1))_3 = 1$ and $((3^{-1}))_4 = 3$ because $((3 \cdot 3))_3 = 1$.

EXAMPLE 7.4.1 A 12-point prime factor algorithm with $N_1 = 3$ and $N_2 = 4$ is as follows. With $A = N_2 = 4$ and $B = N_1 = 3$, and with $C = N_2((N_2^{-1}))_{N_1} = 4$ and $D = N_1((N_1^{-1}))_{N_2} = 9$. Thus, the index mappings for n and k are

$$n = ((4n_1 + 3n_2))_{12} \qquad \begin{cases} 0 \le n_1 \le 2 \\ 0 \le n_2 \le 3 \end{cases}$$

$$k = ((4k_1 + 9k_2))_{12} \qquad \begin{cases} 0 \le k_1 \le 2 \\ 0 \le k_2 \le 3 \end{cases}$$

and the two-dimensional array representation for the input is

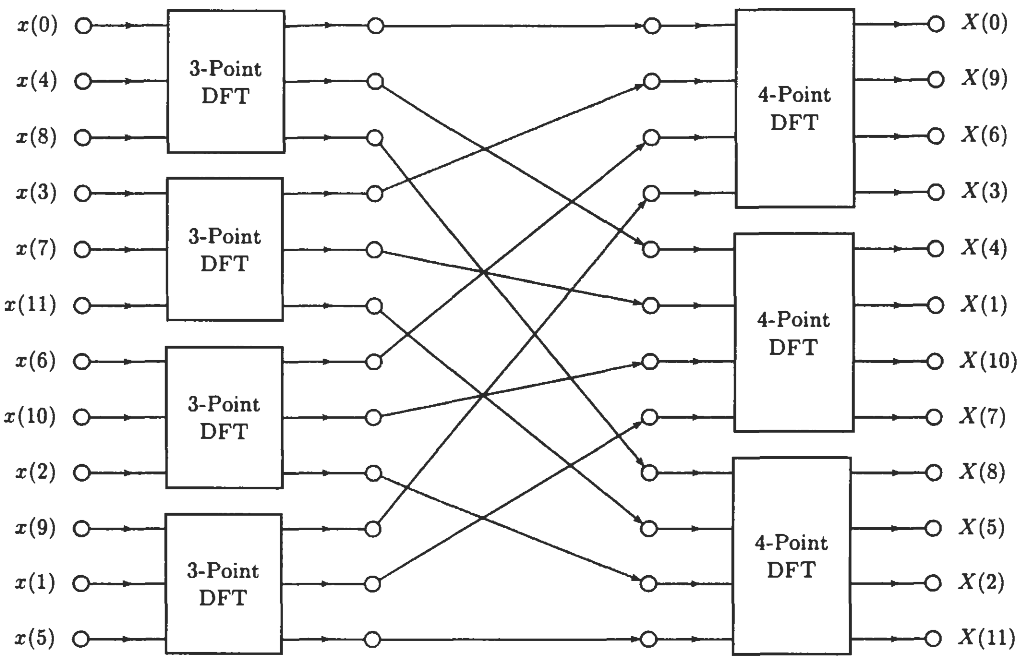
$n_2 \backslash n_1$	0	1	2
0	$x(0)$	$x(4)$	$x(8)$
1	$x(3)$	$x(7)$	$x(11)$
2	$x(6)$	$x(10)$	$x(2)$
3	$x(9)$	$x(1)$	$x(5)$

$k_2 \backslash k_1$	0	1	2
0	$X(0)$	$X(4)$	$X(8)$
1	$X(9)$	$X(1)$	$X(5)$
2	$X(6)$	$X(10)$	$X(2)$
3	$X(3)$	$X(7)$	$X(11)$

The representation for $X(k)$ is therefore

$$X((4k_1 + 9k_2))_{12} = \sum_{n_2=0}^{3}\left\{\left[\sum_{n_1=0}^{2} x((4n_1 + 3n_2))_{12} W_3^{n_1 k_1}\right]\right\} W_4^{n_2 k_2}$$

Thus, the DFT is evaluated by first computing the three-point DFT of each row of the input array, followed by the four-point DFT of each column. The following figure shows how the four-point DFTs are interconnected to the three-point DFTs.



Because a 4-point DFT does not require any multiplications (see Prob. 7.11), and because each 3-point DFT requires only 4 complex multiplications, the 12-point prime factor algorithm requires 16 complex multiplies. For a mixed-radix FFT, there are, in addition, six twiddle factors. The cost for eliminating these six multiplications is an increase in complexity in indexing and in programming.

Solved Problems

Radix-2 FFT Algorithms

7.1 Assume that a complex multiply takes 1 μs and that the amount of time to compute a DFT is determined by the amount of time it takes to perform all of the multiplications.

(*a*) How much time does it take to compute a 1024-point DFT directly?

(*b*) How much time is required if an FFT is used?

(*c*) Repeat parts (*a*) and (*b*) for a 4096-point DFT.

(*a*) Including possible multiplications by ± 1, computing an N-point DFT directly requires N^2 complex multiplications. If it takes 1 μs per complex multiply, the direct evaluation of a 1024-point DFT requires

$$t_{\text{DFT}} = (1024)^2 \cdot 10^{-6}\text{ s} \approx 1.05\text{ s}$$

(*b*) With a radix-2 FFT, the number of complex multiplications is approximately $(N/2)\log_2 N$ which, for $N = 1024$, is equal to 5120. Therefore, the amount of time to compute a 1024-point DFT using an FFT is

$$t_{\text{FFT}} = 5120 \cdot 10^{-6}\text{ms} = 5.12\text{ ms}$$

(*c*) If the length of the DFT is increased by a factor of 4 to $N = 4096$, the number of multiplications necessary to compute the DFT directly increases by a factor of 16. Therefore, the time required to evaluate the DFT directly is

$$t_{\text{DFT}} = 16.78\text{ s}$$

If, on the other hand, an FFT is used, the number of multiplications is

$$2{,}048 \cdot \log_2 4{,}096 = 24{,}576$$

and the amount of time to evaluate the DFT is

$$t_{\text{FFT}} = 24.576\text{ ms}$$

7.2 A complex-valued sequence $x(n)$ of length $N = 8192$ is to be convolved with a complex-valued sequence $h(n)$ of length $L = 512$.

(*a*) Find the number of (complex) multiplications required to perform this convolution directly.

(*b*) Repeat part (*a*) using the overlap-add method with 1024-point radix-2 decimation-in-time FFTs to evaluate the convolutions.

(*a*) If $x(n)$ is of length $N = 8192$, and $h(n)$ of length $L = 512$, performing the convolution directly requires

$$512 \cdot 8,192 = 4,194,304$$

complex multiplications.

(*b*) Using the method of overlap-add with 1024-point FFTs, the number of multiplications is as follows. Because $h(n)$ is of length 512, we may segment $x(n)$ into sequences $x_i(n)$ of length $N = 512$ so that the 1024-point circular convolutions of $h(n)$ with $x_i(n)$ will be the same as linear convolutions (although we could use sections of length 513, this does not result in any computational savings). With the length of $x(n)$ being equal to 8192, this means that we will have 16 sequences of length 512. Therefore, to perform the convolution, we must compute 17 DFTs and 16 inverse DFTs. In addition, we must form the products $Y_i(k) = H(k)X_i(k)$ for $i = 1, 2, \dots, 16$. Thus, the total number of complex multiplications is approximately

$$33 \cdot 512\log_2(1{,}024) + 16 \cdot 1{,}024 = 185{,}344$$

which is about 4.5 percent of the number of complex multiplies necessary to perform the convolution directly.

7.3 Speech that is sampled at a rate of 10 kHz is to be processed in real time. Part of the computations required involve collecting blocks of 1024 speech values and computing a 1024-point DFT and a 1024-point inverse DFT. If it takes 1 μs for each real multiply, how much time remains for processing the data after the DFT and the inverse DFT are computed?

With a 10-kHz sampling rate, a block of 1024 samples is collected every 102.4 ms. With a radix-2 FFT, the number of complex multiplications for a 1024-point DFT is approximately $512 \log_2 1024 = 5120$. With a complex multiply consisting of four real multiplies, this means that we have to perform $5,120 \cdot 4 = 20,480$ real multiplies for the DFT and the same number for the inverse DFT. With 1 μs per multiply, this will take

$$t = 2 \cdot 20.48 = 40.96 \text{ ms}$$

which leaves 61.44 ms for any additional processing.

7.4 Sampling a continuous-time signal $x_a(t)$ for 1 s generates a sequence of 4096 samples.

(*a*) What is the highest frequency in $x_a(t)$ if it was sampled without aliasing?

(*b*) If a 4096-point DFT of the sampled signal is computed, what is the frequency spacing in hertz between the DFT coefficients?

(*c*) Suppose that we are only interested in the DFT samples that correspond to frequencies in the range $200 \leq f \leq 300$ Hz. How many complex multiplies are required to evaluate these values computing the DFT directly, and how many are required if a decimation-in-time FFT is used?

(*d*) How many frequency samples would be needed in order for the FFT algorithm to be more efficient than evaluating the DFT directly?

(*a*) Collecting 4096 samples in 1 s means that the sampling frequency is $f_s = 4096$ Hz. If $x_a(t)$ is to be sampled without aliasing, the sampling frequency must be at least twice the highest frequency in $x_a(t)$. Therefore, $x_a(t)$ should have no frequencies above $f_0 = 2048$ Hz.

(*b*) With a 4096-point DFT, we are sampling $X(e^{j\omega})$ at 4096 equally spaced frequencies between 0 and 2π, which corresponds to 4096 frequency samples over the range $0 \leq f \leq 4096$ Hz. Therefore, the frequency spacing is $\Delta f = 1$ Hz.

(*c*) Over the frequency range from 200 to 300 Hz we have 101 DFT samples. Because it takes 4096 complex multiplies to evaluate each DFT coefficient, the number of multiplies necessary to evaluate only these frequency samples is

$$101 \cdot 4,096 = 413,696$$

On the other hand, the number of multiplications required if an FFT is used is

$$2,048 \log_2 4,096 = 24,576$$

Therefore, even though the FFT generates all of the frequency samples in the range $0 \leq f \leq 4096$ Hz, it is more efficient than evaluating these 101 samples directly.

(*d*) An N-point FFT requires $\frac{1}{2}N \log_2 N$ complex multiplies, and to evaluate M DFT coefficients directly requires $M \cdot N$ complex multiplications. Therefore, the FFT will be more efficient in finding these M samples if

$$M \cdot N > \tfrac{1}{2}N \log_2 N$$

or

$$M \geq \tfrac{1}{2} \log_2 N$$

With $N = 4096$, the number of frequency samples is $M = 6$.

7.5 Because some of the $\frac{1}{2}N \log_2 N$ multiplications in the decimation-in-time and decimation-in-frequency FFT algorithms are multiplications by ± 1, it is possible to more efficiently implement these algorithms by writing programs that specifically excluded these multiplications.

(*a*) How many multiplications are there in an eight-point decimation-in-time FFT if we exclude the multiplications by ± 1?

(*b*) Repeat part (*a*) for a 16-point decimation-in-time FFT.

(*c*) Generalize the results in parts (*a*) and (*b*) for $N = 2^\nu$.

(*a*) For an eight-point decimation-in-time FFT, we may count the number of complex multiplications in the flowgraph given in Fig. 7-6. In the first stage of the FFT, there are no complex multiplications, whereas in the second stage, there are two multiplications by W_8^2. Finally, in the third stage there are three multiplications by W_8, W_8^2, and W_8^3. Thus, there are a total of five complex multiplies.

(*b*) A 16-point DFT is formed from two 8-point DFTs as follows:

$$X(k) = G(k) + W_{16}^k H(k) \qquad k = 0, 1, \ldots, 15$$

where $G(k)$ and $H(k)$ are eight-point DFTs. There are eight butterflies in the last stage that produces $X(k)$ from $G(k)$ and $H(k)$. Because the simplified butterfly in Fig. 7-5(*b*) only requires only one complex multiply, and noting that one of these is by $W_{16}^0 = 1$, we have a total of seven twiddle factors. In addition, we have two 8-point FFTs, which require five complex multiplies each. Therefore, the total number of multiplies is $2 \cdot 5 + 7 = 17$.

(*c*) Let $L(\nu)$ be the number of complex multiplies required for a radix-2 FFT when $N = 2^\nu$. From parts (*a*) and (*b*) we see that $L(3) = 5$ and $L(4) = 17$. Given that an FFT of length $N = 2^{\nu-1}$ requires $L(\nu - 1)$ multiplies, for an FFT of length $N = 2^\nu$, we have an additional $2^{\nu-1}$ butterflies. Because each butterfly requires one multiply, and because one of these multiplies is by $W_N^0 = 1$, the number of multiplies required for an FFT of length 2^ν is

$$L(\nu) = 2 \cdot L(\nu - 1) + 2^{\nu-1} - 1$$

Solving this recursion for $L(\nu)$, we have the following closed-form expression for $L(\nu)$:

$$L(\nu) = 2^\nu\left[\frac{\nu}{2} - 1 + \left(\frac{1}{2}\right)^\nu\right]$$

7.6 The FFT requires the multiplication of complex numbers:

$$(a_1 + jb_1) \cdot (a_2 + jb_2) = c_1 + jd_1$$

(*a*) Write out this complex multiplication, and determine how many real multiplies and real adds are required.

(*b*) Show that the complex multiplication may also be performed as follows:

$$c_1 = (a_1 - b_1) \cdot b_2 + (a_2 - b_2) \cdot a_1$$

$$d_1 = (a_1 - b_1) \cdot b_2 + (a_2 + b_2) \cdot b_1$$

and determine the number of real multiplies and adds required with this method.

(*a*) The product of two complex number is

$$(a_1 + jb_1) \cdot (a_2 + jb_2) = a_1a_2 - b_1b_2 + j(b_1a_2 + a_1b_2)$$

which requires four real multiplies and three real adds.

(*b*) Expanding the expressions for c_1, we have

$$c_1 = (a_1 - b_1) \cdot b_2 + (a_2 - b_2) \cdot a_1 = a_1b_2 - b_1b_2 + a_2a_1 - b_2a_1 = a_1a_2 - b_1b_2$$

as required. Similarly, for d_1 we have

$$d_1 = (a_1 - b_1) \cdot b_2 + (a_2 + b_2) \cdot b_1 = a_1b_2 - b_1b_2 + a_2b_1 + b_2b_1 = a_1b_2 + a_2b_1$$

also as required. This approach only requires three multiplies and four adds.

7.7 The decimation-in-time and decimation-in-frequency FFT algorithms evaluate the DFT of a complex-valued sequence. Show how an N-point FFT program may be used to evaluate the N-point DFT of two *real-valued* sequences.

As we saw in Prob. 6.18, the DFTs of two real-valued sequences may be found from one N-point DFT as follows. First, we form the N-point complex sequence

$$x(n) = x_1(n) + jx_2(n)$$

After finding the N-point DFT of $x(n)$, we extract $X_1(k)$ and $X_2(k)$ from $X(k)$ by exploiting the symmetry of the DFT. Specifically,

$$X_1(k) = \tfrac{1}{2}[X(k) + X^*((N-k))_N]$$

which is the conjugate symmetric part of $X(k)$, and

$$X_2(k) = \tfrac{1}{2}[X(k) - X^*((N-k))_N]$$

which is the conjugate antisymmetric part of $X(k)$.

7.8 Determine how a $2N$-point DFT of a real-valued sequence may be computed using an N-point FFT algorithm.

Let $g(n)$ be a real-valued sequence of length $2N$. From this sequence, we may form two real-valued sequences of length N as follows:

$$x_1(n) = g(2n) \qquad n = 0, 1, \dots, N-1$$
$$x_2(n) = g(2n+1) \qquad n = 0, 1, \dots, N-1$$

From these two sequences, we form the complex sequence

$$x(n) = x_1(n) + jx_2(n)$$

Computing the N-point DFT of $x(n)$, we may then extract the N-point DFTs of $x_1(n)$ and $x_2(n)$ as follows (see Prob. 7.7):

$$X_1(k) = \tfrac{1}{2}[X(k) + X^*((N-k))_N]$$
$$X_2(k) = \tfrac{1}{2}[X(k) - X^*((N-k))_N]$$

Now all that is left to do is to relate the $2N$-point DFT of $g(n)$ to the N-point DFTs $X_1(k)$ and $X_2(k)$. Note that

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

Therefore,

$$G(k) = X_1(k) + W_{2N}^k X_2(k) \qquad k = 0, 1, \dots, 2N-1$$

where the periodicity of $X_1(k)$ and $X_2(k)$ is used to evaluate $G(k)$ for $N < k < 2N$, that is,

$$X_1(k) = X_1(k+N) \qquad X_2(k) = X_2(k+N)$$

7.9 Given an FFT program to find the N-point DFT of a sequence, how may this program be used to find the inverse DFT?

As we saw in Prob. 6.9, we may find $x(n)$ by first using the DFT program to evaluate the sum

$$g(n) = \sum_{n=0}^{N-1} X^*(n)W_N^{nk}$$

which is the DFT of $X^*(k)$. Then, $x(n)$ may be found from $x(n)$ as follows:

$$x(n) = \frac{1}{N} g^*(n)$$

Alternatively, we may find the DFT of $X(k)$,

$$f(n) = \sum_{n=0}^{N-1} X(n) W_N^{nk}$$

and then extract $x(n)$ as follows:

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

7.10 Let $x(n)$ be a sequence of length N with

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

where N is an even integer.

(*a*) Show that the N-point DFT of $x(n)$ has only odd harmonics, that is,

$$X(k) = 0 \qquad k \text{ even}$$

(*b*) Show how to find the N-point DFT of $x(n)$ by finding the $N/2$-point DFT of an appropriately modified sequence.

(*a*) The N-point DFT of $x(n)$ is

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

Because $x(n) = -x(n + N/2)$, if k is even, each term in the sum is zero, and $X(k) = 0$ for $k = 0, 2, 4, \ldots$.

(*b*) In the first stage of a decimation-in-frequency FFT algorithm, we separately evaluate the even-index and odd-index samples of $X(k)$. If $X(k)$ has only odd harmonics, the even samples are zero, and we need only evaluate the odd samples. From Eq. (*7.4*) we see that the odd samples are given by

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

With $x(n) = -x(n + N/2)$ this becomes

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

which is the $N/2$-point DFT of the sequence $y(n) = 2W_N^n x(n)$. Therefore, to find the N-point DFT of $x(n)$, we multiply the first $N/2$ points of $x(n)$ by $2W_N^n$,

$$y(n) = 2W_N^n x(n) \qquad n = 0, 1, 2, \ldots, \frac{N}{2}$$

and then compute the $N/2$-point DFT of $y(n)$. The $N/2$-point DFT of $x(n)$ is then given by

$$X(2k+1) = Y(k) \qquad k = 0, 1, \ldots \frac{(N-2)}{2}$$

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

FFT Algorithms for Composite N

7.11 When the number of points in the DFT is a power of 4, we can use a radix-2 FFT algorithm. However, when $N = 4^\nu$, it is more efficient to use a radix-4 FFT algorithm.

(*a*) Derive the radix-4 decimation-in-time FFT algorithm when $N = 4^\nu$.

(*b*) Draw the structure for the butterfly in the radix-4 FFT, and compare the number of complex multiplies and adds with a radix-4 FFT to a radix-2 FFT.

(*a*) To derive a decimation-in-time radix-4 FFT, let $N_1 = N/4$ and $N_2 = 4$, and define the index maps

$$n = 4 \cdot n_1 + n_2 \qquad \begin{cases} 0 \le n_1 \le \dfrac{N}{4} - 1 \\ 0 \le n_2 \le 4 \end{cases}$$

$$k = k_1 + \frac{N}{4} \cdot k_2 \qquad \begin{cases} 0 \le k_1 \le \dfrac{N}{4} - 1 \\ 0 \le k_2 \le 3 \end{cases}$$

We then express $X(k)$ using the decomposition given in Eq. (7.7) with $N_1 = N/4$ and $N_2 = 4$,

$$X(k) = X\left(k_1 + \frac{N}{4}k_2\right) = \sum_{n_2=0}^{3}\left\{\left[\sum_{n_1=0}^{\frac{N}{4}-1} x(4n_1 + n_2)W_{N/4}^{n_1k_1}\right]W_N^{k_1n_2}\right\}W_4^{k_2n_2}$$

The inner summation,

$$G(n_2, k_1) = \sum_{n_1=0}^{\frac{N}{4}-1} x(4n_1 + n_2)W_{N/4}^{n_1k_1}$$

is the $N/4$-point DFT of the sequence $x(4n_1 + n_2)$, and the outer summation is a 4-point DFT,

$$X\left(k_1 + \frac{N}{4}k_2\right) = \sum_{n_2=0}^{3} \tilde{G}(k_1, n_2)W_4^{k_2n_2}$$

where $$\tilde{G}(n_2, k_1) = W_N^{k_1n_2}G(n_2, k_1)$$

Since $W_4 = -j$, these 4-point transforms have the form

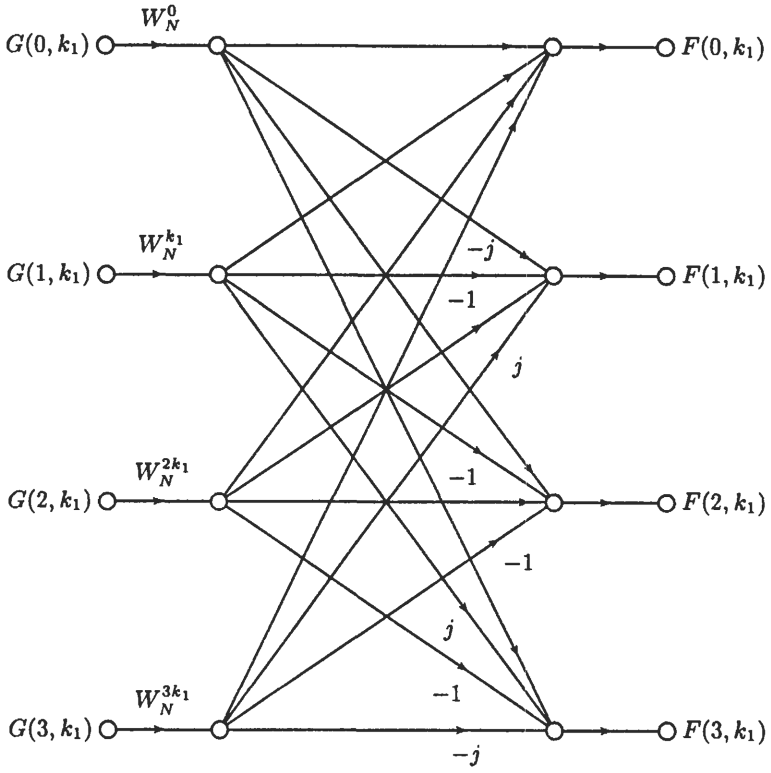
$$X\left(k_1 + \frac{N}{4}k_2\right) = \tilde{G}(k_1, 0) + (-j)^{k_1}\tilde{G}(1, k_1) + (-1)^{k_1}\tilde{G}(2, k_1) + (j)^{k_1}\tilde{G}(3, k_1)$$

for $k_1 = 0, 1, 2, 3$, and $n_2 = 0, 1, \ldots, (N/4) - 1$. If $N_2 = N/4$ is divisible by 4, then the process is repeated. In this way, we generate $\nu = \log_4 N$ stages with $N/4$ butterflies in each stage.

(*b*) The 4-point butterflies in the radix-4 FFT perform operations of the form

$$F(k_2, k_1) = \sum_{n_2=0}^{3}\left[G(n_2, k_1)W_N^{k_1n_2}\right]W_4^{k_2n_2} \qquad k_2 = 0, 1, 2, 3$$

With $W_4^{k_2 n_2} = j^{k_2 n_2}$, the butterflies have the structure shown in the figure below.



Since multiplications by $\pm j$ only requires interchanging real and imaginary parts and possibly changing a sign bit, then each 4-point butterfly only requires 3 complex multiplications. With $\nu = \log_4 N$ stages, and $N/4$ butterflies per stage, the number of complex multiplies for a DFT of length $N = 4^\nu$ is

$$3 \cdot \frac{N}{4} \log_4 N = \frac{3N}{8} \log_2 N$$

For a radix-2 decimation-in-time FFT, on the other hand, the number of multiplications is

$$\frac{N}{2} \log_2 N$$

Therefore, the number of multiplications in a radix-4 FFT is $\frac{3}{16}$ times the number in a radix-2 FFT.

7.12 Suppose that we would like to find the N-point DFT of a sequence where N is a power of 3, $N = 3^\nu$.

(*a*) Develop a radix-3 decimation-in-time FFT algorithm, and draw the corresponding flowgraph for $N = 9$.

(*b*) How many multiplications are required for a radix-3 FFT?

(*c*) Can the computations be performed in place?

(*a*) A radix-3 decimation-in-time FFT may be derived in exactly the same way as a radix-2 FFT. First, $x(n)$ is decimated by a factor of 3 to form three sequences of length $N/3$:

$$f(n) = x(3n) \qquad n = 0, 1, \ldots, \frac{N}{3} - 1$$

$$g(n) = x(3n+1) \qquad n = 0, 1, \ldots, \frac{N}{3} - 1$$

$$h(n) = x(3n+2) \qquad n = 0, 1, \ldots, \frac{N}{3} - 1$$

Expressing the N-point DFT in terms of these sequences, we have

$$X(k) = \sum_{n=0,3,6,\dots} x(n)W_N^{nk} + \sum_{n=1,4,5,\dots} x(n)W_N^{nk} + \sum_{n=2,5,7,\dots} x(n)W_N^{nk}$$

$$= \sum_{l=0}^{\frac{N}{3}-1} f(l)W_N^{3lk} + \sum_{l=0}^{\frac{N}{3}-1} g(l)W_N^{(3l+1)k} + \sum_{l=0}^{\frac{N}{3}-1} h(l)W_N^{(3l+2)k}$$

Since $W_N^{3lk} = W_{N/3}^{lk}$, then

$$X(k) = \sum_{l=0}^{\frac{N}{3}-1} f(l)W_{N/3}^{lk} + W_N^k \sum_{l=0}^{\frac{N}{3}-1} g(l)W_{N/3}^{lk} + W_N^{2k} \sum_{l=0}^{\frac{N}{3}-1} h(l)W_{N/3}^{lk}$$

Note that the first term is the $N/3$-point DFT of $f(n)$, the second is W_N^k times the $N/3$-point DFT of $g(n)$, and the third is W_N^{2k} times the $N/3$-point DFT of $h(n)$,

$$X(k) = F(k) + W_N^k G(k) + W_N^{2k} H(k)$$

We may continue decimating by factors of 3 until we are left with only 3-point DFTs. The flowgraph for a 9-point decimation-in-time FFT is shown in Fig. 7-11. Only one of the 3-point butterflies is shown in the second stage in order to allow for the labeling of the branches. The complete flowgraph is formed by replicating this 3-point butterfly up by one node, and down by one node, and changing the branch multiplies to their appropriate values.

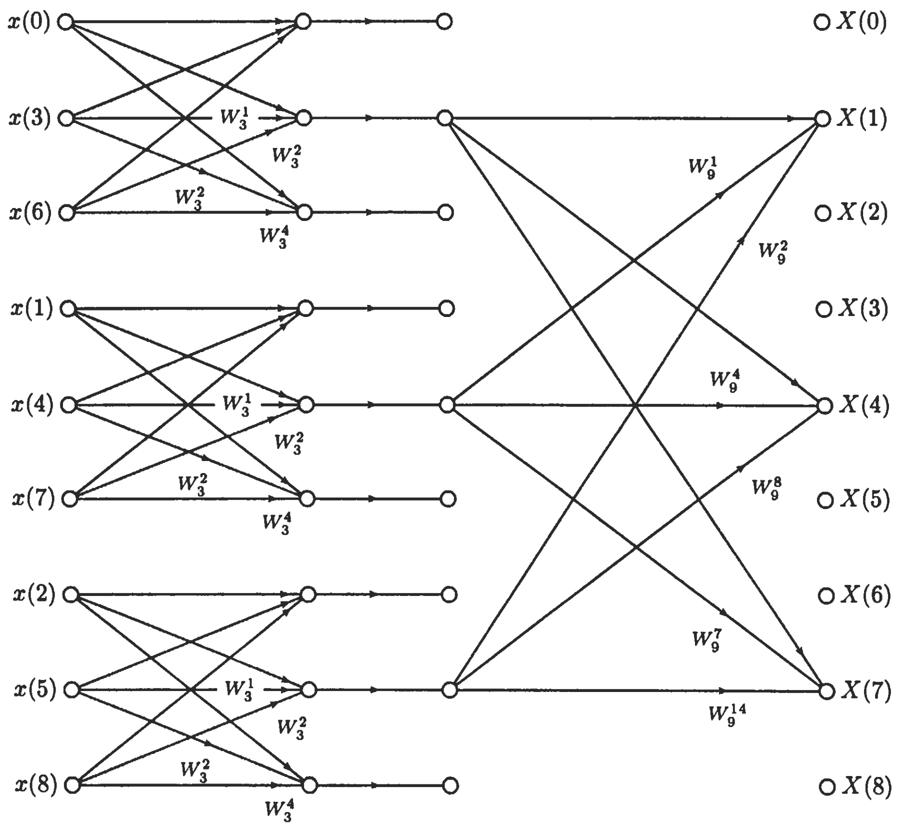


Fig. 7-11. Flowgraph for a 9-point decimation-in-time FFT (only one butterfly in the second stage is shown).

(*b*) If $N = 3^\nu$, then there are ν stages in the radix-3 FFT. The general form of each 3-point butterfly, shown in the second stage of the flowgraph in Fig. 7-11, requires six multiplies (some require fewer if we do not consider multiplications by ± 1). Since there are $N/3$ butterflies in each stage, then the total number of multiplications is

$$6N \log_3 N$$

(*c*) Yes, the computations may be performed in place.

7.13 Derive a radix-3 decimation-in-frequency FFT for $N = 3^\nu$, and draw the corresponding flowgraph for $N = 9$.

As with the radix-2 decimation-in-frequency FFT, with $N = 3^\nu$, we separately evaluate the indices for which $((k))_3 = 0$, $((k))_3 = 1$, and $((k))_3 = 2$. For $((k))_3 = 0$ (i.e., k is a multiple of 3),

$$X(3k) = \sum_{n=0}^{N-1} x(n) W_N^{3nk}$$

Separating this sum into the first $N/3$ points, the second $N/3$ points, and the last $N/3$ points, and using the fact that $W_N^{3nk} = W_{N/3}^{nk}$, this becomes

$$X(3k) = \sum_{n=0}^{\frac{N}{3}-1} x(n) W_{N/3}^{nk} + \sum_{n=\frac{N}{3}}^{\frac{2N}{3}-1} x(n) W_{N/3}^{nk} + \sum_{n=\frac{2N}{3}}^{N-1} x(n) W_{N/3}^{nk}$$

With a change in the indexing in the second and third sums, we have

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

Finally, because $W_{N/3}^{n+\frac{N}{3}} = W_{N/3}^{n}$, and $W_{N/3}^{n+\frac{2N}{3}} = W_{N/3}^{n}$,

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

which is the $N/3$-point DFT of the sequence in brackets.

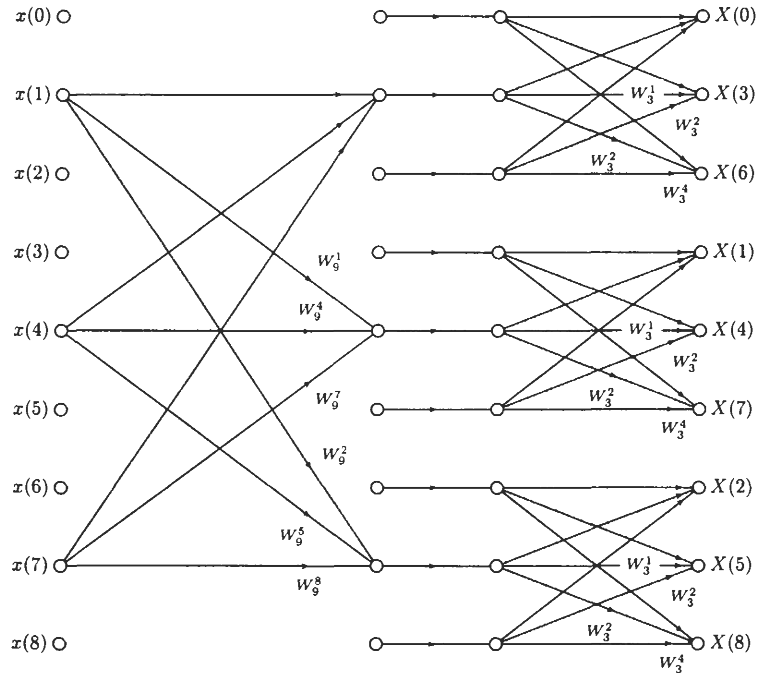
Proceeding in the same way for the samples $X(3k+1)$, we have

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

Finally, for the samples $X(3k+2)$ we have

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

The flowgraph for a nine-point decimation-in-frequency FFT is shown below.



7.14 Suppose that we have a number of eight-point decimation-in-time FFT chips. How could these chips be used to compute a 24-point DFT?

A 24-point DFT is defined by

$$X(k) = \sum_{n=0}^{23} x(n) W_{24}^{nk}$$

Decimating $x(n)$ by a factor of 3, we may decompose this DFT into three 8-point DFTs as follows:

$$\begin{aligned} X(k) &= \sum_{n=0}^{7} x(3n) W_{24}^{3nk} + \sum_{n=0}^{7} x(3n+1) W_{24}^{(3n+1)k} + \sum_{n=0}^{7} x(3n+2) W_{24}^{(3n+2)k} \\ &= \sum_{n=0}^{7} x(n) W_8^{nk} + W_{24}^{k} \sum_{n=0}^{7} x(3n+1) W_8^{nk} + W_{24}^{2k} \sum_{n=0}^{7} x(3n+2) W_8^{nk} \end{aligned}$$

Therefore, if we form the three sequences

$$\begin{aligned} f(n) &= x(3n) & n &= 0, 1, 2, \dots, 7 \\ g(n) &= x(3n+1) & n &= 0, 1, 2, \dots, 7 \\ h(n) &= x(3n+2) & n &= 0, 1, 2, \dots, 7 \end{aligned}$$

and use the 8-point FFT chips to find the DFTs $F(k)$, $G(k)$, and $H(k)$, the 24-point DFT of $x(n)$ may be found by combining the outputs of the 8-point FFTs as follows:

$$X(k) = F(k) + W_{24}^{k} G(k) + W_{24}^{2k} H(k)$$

Prime Factor FFT

7.15 Find the index maps for a 21-point prime factor FFT with $N_1 = 7$ and $N_2 = 3$. How many multiplications are required compared to a 32-point radix-2 decimation-in-time FFT?

For a 21-point prime factor FFT with $N_1 = 7$ and $N_2 = 3$, we set $A = N_2 = 3$ and $B = N_1 = 7$. Then, with $C = N_2((N_2^{-1}))_{N_1} = 15$ and $D = N_1((N_1^{-1}))_{N_2} = 7$, we have the following index mappings:

$$n = ((3n_1 + 7n_2))_N \qquad \begin{cases} 0 \le n_1 \le 6 \\ 0 \le n_2 \le 2 \end{cases}$$

$$k = ((15k_1 + 7k_2))_N \qquad \begin{cases} 0 \le k_1 \le 6 \\ 0 \le k_2 \le 2 \end{cases}$$

Thus, the two-dimensional array representation for the input is

$n_2 \backslash n_1$	0	1	2	3	4	5	6
0	$x(0)$	$x(3)$	$x(6)$	$x(9)$	$x(12)$	$x(15)$	$x(18)$
1	$x(7)$	$x(10)$	$x(13)$	$x(16)$	$x(19)$	$x(1)$	$x(4)$
2	$x(14)$	$x(17)$	$x(20)$	$x(2)$	$x(5)$	$x(8)$	$x(11)$

and the two-dimensional array for the output is

$k_2 \backslash k_1$	0	1	2	3	4	5	6
0	$X(0)$	$X(15)$	$X(9)$	$X(3)$	$X(18)$	$X(12)$	$X(6)$
1	$X(7)$	$X(1)$	$X(16)$	$X(10)$	$X(4)$	$X(19)$	$X(13)$
2	$X(14)$	$X(8)$	$X(2)$	$X(17)$	$X(11)$	$X(5)$	$X(20)$

With the prime factor FFT, there are no twiddle factors. Therefore, the only multiplications necessary are those required to compute the three 7-point DFTs, and the seven 3-point DFTs. Because each 3-point DFT requires 6 complex multiplies, and each 7-point DFT requires 42, the number of multiplies for a 21-point prime factor FFT is $(7)(6) + (3)(42) = 168$. For a 32-point radix-2 FFT, on the other hand, we require

$$16 \log_2 32 = 80$$

complex multiplies. Therefore, it would be more efficient to pad a 21-point sequence with zeros and compute a 32-point DFT. The increased efficiency is a result of the fact that $32 = 2^5$ is a much more composite number than $21 = 7 \cdot 3$.

7.16 Suppose that we would like to compute a 15-point DFT of a sequence $x(n)$.

(*a*) Using a mixed-radix FFT with $N_1 = 5$ and $N_2 = 3$, the DFT is decomposed into two stages, with the first consisting of three 5-point DFTs, and the second stage consisting of five 3-point DFTs. Make a sketch of the connections between the five- and three-point DFTs, indicating any possible twiddle factors, and the order of the inputs and outputs.

(*b*) Repeat part (*a*) for the prime factor algorithm with $N_1 = 5$ and $N_2 = 3$, and determine how many complex multiplies are saved with the prime factor algorithm.

(*a*) Using a mixed-radix FFT with $N_1 = 5$ and $N_2 = 3$, the index mappings for n and k are as follows:

$$n = 3n_1 + n_2 \qquad \begin{cases} 0 \le n_1 \le 4 \\ 0 \le n_2 \le 2 \end{cases}$$

$$k = k_1 + 5k_2 \qquad \begin{cases} 0 \le k_1 \le 4 \\ 0 \le k_2 \le 2 \end{cases}$$

Thus, the two-dimensional array representation for the input is

$n_2 \backslash n_1$	0	1	2	3	4
0	$x(0)$	$x(3)$	$x(6)$	$x(9)$	$x(12)$
1	$x(1)$	$x(4)$	$x(7)$	$x(10)$	$x(13)$
2	$x(2)$	$x(5)$	$x(8)$	$x(11)$	$x(14)$

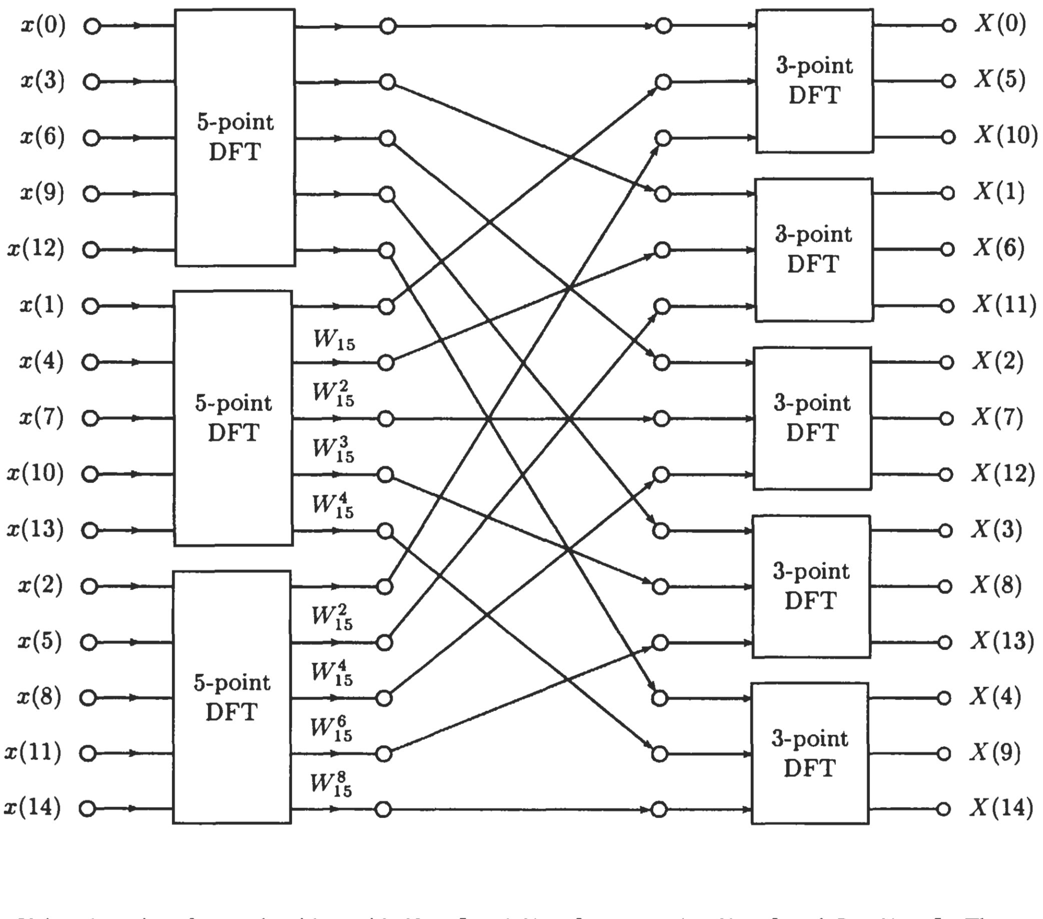
After the five-point DFT of each row in the data array is computed, the resulting complex array is multiplied by the array of twiddle factors:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & W_{15} & W_{15}^2 & W_{15}^3 & W_{15}^4 \\ 1 & W_{15}^2 & W_{15}^4 & W_{15}^6 & W_{15}^8 \end{bmatrix}$$

The last step then involves computing the three-point DFT of each column. This produces the output array $X(k)$, which is

k_2 \ k_1	0	1	2	3	4
0	$X(0)$	$X(1)$	$X(2)$	$X(3)$	$X(4)$
1	$X(5)$	$X(6)$	$X(7)$	$X(8)$	$X(9)$
2	$X(10)$	$X(11)$	$X(12)$	$X(13)$	$X(14)$

The connections between the three- and five-point DFTs are shown in the following figure, along with the eight twiddle factors:



(*b*) Using the prime factor algorithm with $N_1 = 5$ and $N_2 = 3$, we set $A = N_2 = 3$ and $B = N_1 = 5$. Then, with $C = N_2((N_2^{-1}))_{N_1} = 6$ and $D = N_1((N_1^{-1}))_{N_2} = 10$, we have the following index mappings for n and k:

$$n = ((3n_1 + 5n_2))_{15} \qquad \begin{cases} 0 \le n_1 \le 4 \\ 0 \le n_2 \le 2 \end{cases}$$

$$k = ((6k_1 + 10k_2))_{15} \qquad \begin{cases} 0 \le k_1 \le 4 \\ 0 \le k_2 \le 2 \end{cases}$$

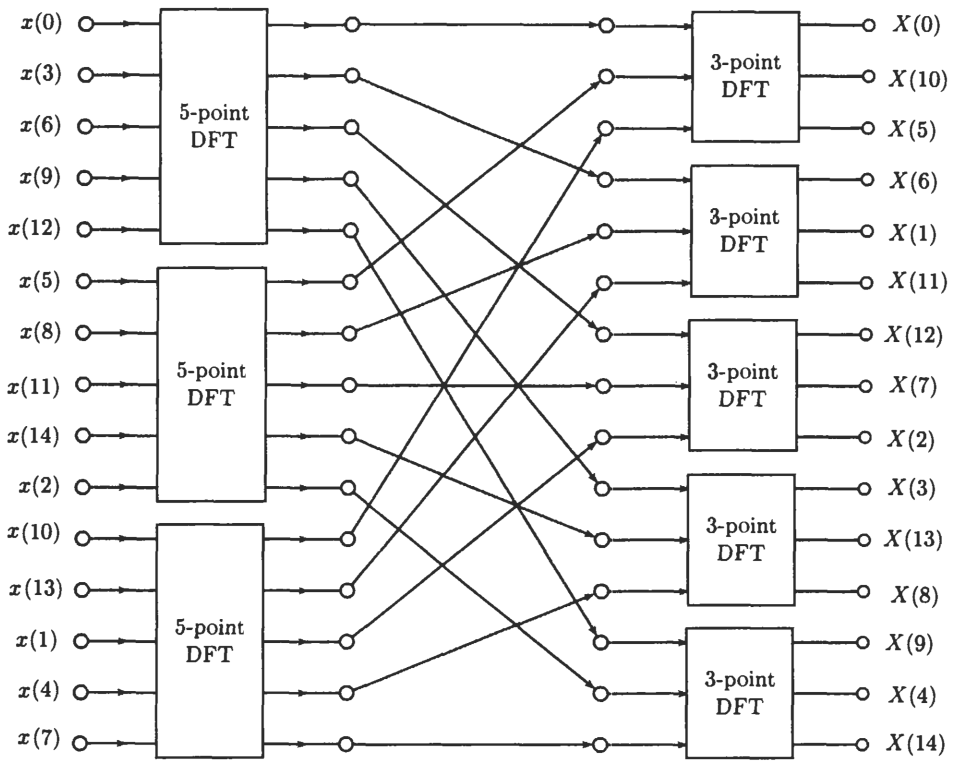
The two-dimensional array representation for the input is

$n_2 \backslash n_1$	0	1	2	3	4
0	$x(0)$	$x(3)$	$x(6)$	$x(9)$	$x(12)$
1	$x(5)$	$x(8)$	$x(11)$	$x(14)$	$x(2)$
2	$x(10)$	$x(13)$	$x(1)$	$x(4)$	$x(7)$

and for the output array we have

$k_2 \backslash k_1$	0	1	2	3	4
0	$X(0)$	$X(6)$	$X(12)$	$X(3)$	$X(9)$
1	$X(10)$	$X(1)$	$X(7)$	$X(13)$	$X(4)$
2	$X(5)$	$X(11)$	$X(2)$	$X(8)$	$X(14)$

The interconnections between the five- and three-point DFTs are the same as in the mixed-radix algorithm. However, there are no twiddle factors, and the ordering of the input and output arrays is different. The 15-point prime factor algorithm is diagrammed in the figure below.



The savings with the prime factor algorithm over the mixed-radix FFT are the eight complex multiplies by the twiddle factors.

Supplementary Problems

Radix-2 FFT Algorithms

7.17 Let $x(n)$ be a sequence of length 1024 that is to be convolved with a sequence $h(n)$ of length L. For what values of L is it more efficient to perform the convolution directly than it is to perform the convolution by taking the inverse DFT of the product $X(k)H(k)$ and evaluating the DFTs using a radix-2 FFT algorithm?

7.18 Suppose that we have a 1025-point data sequence (1 more than $N = 2^{10}$). Instead of discarding the final value, we zero pad the sequence to make it of length $N = 2^{11}$ so that we can use a radix-2 FFT algorithm. (*a*) How many multiplications and additions are required to compute the DFT using a radix-2 FFT algorithm? (*b*) How many multiplications and additions would be required to compute a 1025-point DFT directly?

FFT Algorithms for Composite N

7.19 In a radix-3 decimation-in-time FFT, how is the input sequence indexed?

7.20 How many complex multiplications are necessary in a radix-3 decimation-in-frequency FFT?

7.21 Consider the FFT algorithm given in Example 7.3.2. (*a*) How many multiplications and additions are required to compute a 12-point DFT? (*b*) How many multiplications and additions are necessary if the 12-point DFT is computed directly?

Prime Factor FFT

7.22 Find the index maps for a 99-point prime factor FFT with $N_1 = 11$ and $N_2 = 9$.

7.23 How many complex multiplications are required for a 12-point prime factor FFT with $N_1 = 4$ and $N_3 = 3$ if we do not count multiplications by ± 1 and $\pm j$?

7.24 How many twiddle factors are there in a 99-point prime factor FFT with $N_1 = 11$ and $N_2 = 9$?

7.25 How many complex multiplications are required for a 15-point prime factor FFT if we do not count multiplications by ± 1?

Answers to Supplementary Problems

7.17 $L < 33$.

7.18 (*a*) 11,264. (*b*) 1,050,625.

7.19 The index for $x(n)$ is expressed in ternary form, and then the ternary digits are read in reverse order.

7.20 The same as a decimation-in-time FFT, which is $2N \log_3 N$.

7.21 (*a*) Each 4-point DFT requires no multiplies and 12 adds, and each 3-point DFT requires 6 multiplies and 6 adds. With 6 twiddle factors, there are $6 + (4)(6) = 30$ multiplies and $(4)(6) + (3)(12) = 60$ adds. (*b*) 144 multiplies and 132 adds.

7.22 $n = 9n_1 + 11n_2$, and $k = 45k_1 + 55k_2$.

7.23 24.

7.24 None.

7.25 90.