Chapter

8

Discrete Fourier Transform

The *Fourier series* (FS), introduced in Chap. 1, links the continuous-time domain to the discrete-frequency domain; and the *Fourier transform* (FT) links the continuous-time domain to the continuous-frequency domain. The *discrete-time Fourier transform* (DTFT), introduced in Sec. 7.2, links the discrete-time domain to the continuous-frequency domain. In this chapter, we examine the *discrete Fourier transform* (DFT) which links the discrete-time and discrete-frequency domains. A complete treatment of the design and coding of DFT algorithms can fill volumes (see Brigham 1975; Burrus and Parks 1984; Nussbaumer 1981). Rather than attempt complete coverage of DFTs, this chapter presents only those aspects that are germane to the design of digital filters. Coverage of the so-called fast algorithms for implementation of the DFT is limited to one specific type of algorithm along with an examination of the computational savings that fast algorithms can provide.

8.1 Discrete Fourier Transform

The discrete Fourier transform and its inverse are given by

$$X[m] = \sum_{n=0}^{N-1} x[n]\, e^{-j2\pi mnFT} \qquad m = 0, 1, \ldots, N-1 \tag{8.1a}$$

$$= \sum_{n=0}^{N-1} x[n] \cos(2\pi mnFT) + j \sum_{n=0}^{N-1} x[n] \sin(2\pi mnFT) \tag{8.1b}$$

$$x[n] = \sum_{m=0}^{N-1} X[m]\, e^{j2\pi mnFT} \qquad n = 0, 1, \ldots, N-1 \tag{8.2a}$$

$$= \sum_{m=0}^{N-1} X[m] \cos(2\pi mnFT) + j \sum_{m=0}^{N-1} X[m] \sin(2\pi mnFT) \tag{8.2b}$$

It is a common practice in the DSP literature to "bury the details" of Eqs.

(8.1) and (8.2) by defining $W_N = e^{j2\pi/N} = e^{j2\pi FT}$ and rewriting (8.1*a*) and (8.2*a*) as

$$X[m] = \sum_{n=0}^{N-1} x[n]\, W_N^{-mn} \tag{8.3}$$

$$x[n] = \sum_{m=0}^{N-1} X[m]\, W_N^{mn} \tag{8.4}$$

Since the exponents in (8.3) and (8.4) differ only in sign, another common practice in writing DFT software is to write only a single routine that can evaluate either (8.3) or (8.4) depending upon the value of an input flag being equal to $+1$ or -1. Back in the "olden days," when memory and disk space were expensive, this was a big deal; but these days, having two separate routines may pay for itself in terms of clarity, execution speed, and simplified calling sequences.

Parameter selection

In designing a DFT for a particular application, values must be chosen for the parameters N, T, and F. N is the number of time sequence values $x[n]$ over which the DFT summation is performed to compute each frequency sequence value. It is also the total number of frequency sequence values $X[m]$ produced by the DFT. For convenience, the complete set of N consecutive time sequence values is referred to as the *input record*, and the complete set of N consecutive frequency sequence values is called the *output record*. T is the time interval between two consecutive samples of the input sequence, and F is the frequency interval between two consecutive samples of the output sequence. The selection of values for N, F, and T is subject to the following constraints, which are a consequence of the sampling theorem and the inherent properties of the DFT:

1. The inherent properties of the DFT require that $FNT = 1$.
2. The sampling theorem requires that $T \leq 1/(2f_H)$, where f_H is the highest significant frequency component in the continuous-time signal.
3. The record length in time is equal to NT or $1/F$.
4. Many fast DFT algorithms (such as the one discussed in Sec. 8.5) require that N be an integer power of 2.

Example 8.1 Choose values of N, F, and T given that F must be 5 Hz or less, N must be an integer power of 2, and the bandwidth of the input signal is 300 Hz. For the values chosen, determine the longest signal that can fit into a single input record.

solution From constraint 2 above, $T \leq 1/(2f_H)$. Since $f_H = 300$ Hz, $T \leq 1.66$ ms. If we select $F = 5$ and $T = 0.0016$, then $N \geq 125$. Since N must be an integer power of 2, then we choose $N = 128 = 2^7$, and F becomes 4.883 Hz. Using these values, the input record will span $NT = 204.8$ ms.

Example 8.2 Assuming that $N = 256$ and F must be 5 Hz or less, determine the highest input-signal bandwidth that can be accommodated without aliasing.

solution Since $FNT = 1$, then $T \geq 781.25\ \mu s$. This corresponds to a maximum f_H of 640 Hz.

Periodicity

A periodic function of time will have a discrete-frequency spectrum, and a discrete-time functon will have a spectrum that is periodic. Since the DFT relates a discrete-time function to a corresponding discrete-frequency function, this implies that both the time function and frequency function are periodic as well as discrete. This means that some care must be exercised in selecting DFT parameters and in interpreting DFT results, but it does not mean that the DFT can be used only on periodic digital signals. Based on the DFT's inherent periodicity, it is a common practice to regard the points from $n = 1$ through $n = N/2$ as positive and the points from $n = N/2$ through $n = N - 1$ as negative. Since both the time and frequency sequences are periodic, the values at points $n = N/2$ through $n = N - 1$ are in fact equal to the values at points $n = N/2$ through $n = -1$. Under this convention, it is convenient to redefine the concept of even and odd sequences: If $x[N - n] = x[n]$, the $x[n]$ is even symmetric, and if $x[N - n] = -x[n]$, then $x[n]$ is odd symmetric or antisymmetric.

8.2 Properties of the DFT

The DFT exhibits a number of useful properties and operational relationships that are similar to the properties of the continuous Fourier transform discussed in Chap. 1.

Linearity

The DFT relating $x[n]$ and $X[m]$:

$$x[n] \underset{\text{IDFT}}{\overset{\text{DFT}}{\rightleftharpoons}} X[m]$$

is homogeneous

$$a\,X[n] \underset{\text{IDFT}}{\overset{\text{DFT}}{\rightleftharpoons}} a\,X[m]$$

additive

$$x[n] + y[n] \underset{\text{IDFT}}{\overset{\text{DFT}}{\rightleftharpoons}} X[m] + Y[m]$$

and therefore linear

$$a\,x[n] + b\,y[n] \underset{\text{IDFT}}{\overset{\text{DFT}}{\rightleftharpoons}} a\,X[m] + b\,Y[m]$$

Symmetry

A certain symmetry exists between a time sequence and the corresponding frequency sequence produced by the DFT. Given that $x[n]$ and $X[m]$ constitute a DFT pair, that is,

$$x[n] \underset{\text{IDFT}}{\overset{\text{DFT}}{\rightleftharpoons}} X[m]$$

then

$$\frac{1}{N} X[n] \underset{\text{IDFT}}{\overset{\text{DFT}}{\rightleftharpoons}} x[-m]$$

Time shifting

A time sequence $x[n]$ can be shifted in time by subtracting an integer from n. Shifting the time sequence will cause the corresponding frequency sequence to be phase-shifted. Specifically, given

$$x[n] \underset{\text{IDFT}}{\overset{\text{DFT}}{\rightleftharpoons}} X[m]$$

then

$$x[n-k] \underset{\text{IDFT}}{\overset{\text{DFT}}{\rightleftharpoons}} X[m]\, e^{-j2\pi mk/N}$$

Frequency shifting

Time sequence modulation is accomplished by multiplying the time sequence by an imaginary exponential term $e^{j2\pi nk/N}$. This will cause a frequency shift of the corresponding spectrum. Specifically, given

$$x[n] \underset{\text{IDFT}}{\overset{\text{DFT}}{\rightleftharpoons}} X[m]$$

then

$$x[n]\, e^{j2\pi mk/N} \underset{\text{IDFT}}{\overset{\text{DFT}}{\rightleftharpoons}} X[m-k]$$

Even and odd symmetry

Consider a time sequence $x[n]$ and the corresponding frequency sequence $X[m] = X_R[m] + jX_I[m]$, where $X_R[m]$ and $X_I[m]$ are real valued. If $x[n]$ is even, then $X[m]$ is real valued and even:

$$x[-n] = x[n] \Leftrightarrow X[m] = X_R[m] = X_R[-m]$$

If $x[n]$ is odd, then $X[m]$ is imaginary and odd:

$$x[-n] = -x[n] \Leftrightarrow X[m] = jX_I[m] = -jX_I[-m]$$

Real and imaginary properties

In general, the DFT of a real-valued time sequence will have an even real component and an odd imaginary component. Conversely, an imaginary-

valued time sequence will have an odd real component and an even imaginary component. Given a time sequence $x[n] = x_R[n] + jx_I[n]$ and the corresponding frequency sequence $X[m] = X_R[m] + jX_I[m]$, then

$$x[n] = x_R[n] \Leftrightarrow X_R[m] = X_R[-m] \qquad X_I[m] = -X_I[-m]$$

$$x[n] = jx_I[n] \Leftrightarrow X_R[m] = -X_R[-m] \qquad X_I[m] = X_I[-m]$$

8.3 Implementing the DFT

The C function shown in Listing 8.1 is the "brute-force" implementation of Eq. (8.1). This function is an example of grossly inefficient code. The sine and cosine operations are each performed N^2 times to compute an N-point DFT. Since

$$\exp\left(\frac{-2\pi jk}{N}\right) = \exp\left[\frac{-2\pi j(k \bmod N)}{N}\right] \tag{8.5}$$

it follows that there are only N different values of **phi** that need be computed in **dft( )**. We can trade space for speed by precomputing and storing the values of **sin(phi)** and **cos(phi)** for **phi** $= 0, 1, \ldots, N-1$. The resulting modified function **dft2( )** is presented in Listing 8.2.

8.4 Fast Fourier Transforms

Consider the operation of **dft2( )** for the case of $N = 8$. The computation of **sumRe** involves the product of **x[n].Re** and **cosVal[k]** for $n = 0, 1, \ldots, 7$. For any given value of n, the value of k is determined by the value of m using

$$k = mn \text{ modulo } N$$

For $N = 8$, there are 64 possible combinations of (m, n) and only 8 possible values of k. Obviously, more than 1 combination of (m, n) will map into each value of k as indicated in Table 8.1.

TABLE 8.1 Values of k as a Function of (m, n) for an 8-point DFT

n	$k(0, n)$	$k(1, n)$	$k(2, n)$	$k(3, n)$	$k(4, n)$	$k(5, n)$	$k(6, n)$	$k(7, n)$
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	1	4	7	2	5
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	5	4	3	2	1

For $N = 8$, the function **dft2( )** computes the product **x[0].Re*cosVal[0]** a total of eight times—once for each different value of m. Similarly, the product **x[4].Re*cosVal[0]** is computed a total of four times—once for each odd value of m. A variety of different fast DFT algorithms has been developed by reordering and regrouping hhe DFT computations so as to minimize or eliminate the need for multiple calculation of the same product.

The expanded equations for computation of $X(0)$ through $X(7)$ for an 8-point DFT are listed in Table 8.2. Making use of Eq. (8.5) along with the commutative, associative, and distributive properties of addition and multiplication, the equations of Table 8.2 can be rewritten in the form shown in Table 8.3. Examination of these equations reveals that they share many common terms that can be computed once and then used as needed without having to be computed over again. Use of these common terms is easier to understand if the equations are presented in the form of a signal flow graph as in Fig. 8.1. The format of this signal flow graph has been slightly modified from the format of Sec. 7.6 in order to reduce the clutter somewhat. In the modified format, each circle represents one (possibly complex) addition and one (possibly complex) multiplication. The term corresponding to the line with the arrowhead entering the circle is multiplied by the constant within the circle and then added to the term correesponding to the other line entering the circle. The notation W^n represents $\exp(-j2\pi/N)$. The computa-

TABLE 8.2 Equations for Computation of an 8-point DFT

$$X(0) = x(0)W^0 + x(1)W^0 + x(2)W^0 + x(3)W^0 + x(4)W^0 + x(5)W^0 + x(6)W^0 + x(7)W^0$$
$$X(1) = x(0)W^0 + x(1)W^1 + x(2)W^2 + x(3)W^3 + x(4)W^4 + x(5)W^5 + x(6)W^6 + x(7)W^7$$
$$X(2) = x(0)W^0 + x(1)W^2 + x(2)W^4 + x(3)W^6 + x(4)W^8 + x(5)W^{10} + x(6)W^{12} + x(7)W^{14}$$
$$X(3) = x(0)W^0 + x(1)W^3 + x(2)W^6 + x(3)W^9 + x(4)W^{12} + x(5)W^{15} + x(6)W^{18} + x(7)W^{21}$$
$$X(4) = x(0)W^0 + x(1)W^4 + x(2)W^8 + x(3)W^{12} + x(4)W^{16} + x(5)W^{20} + x(6)W^{24} + x(7)W^{28}$$
$$X(5) = x(0)W^0 + x(1)W^5 + x(2)W^{10} + x(3)W^{15} + x(4)W^{20} + x(5)W^{25} + x(6)W^{30} + x(7)W^{35}$$
$$X(6) = x(0)W^0 + x(1)W^6 + x(2)W^{12} + x(3)W^{18} + x(4)W^{24} + x(5)W^{30} + x(6)W^{36} + x(7)W^{42}$$
$$X(7) = x(0)W^0 + x(1)W^7 + x(2)W^{14} + x(3)W^{21} + x(4)W^{28} + x(5)W^{35} + x(6)W^{42} + x(7)W^{49}$$

TABLE 8.3 Factored Equations for Computation of an 8-point DFT

$$X(0) = \{[x(0) + x(4)W^0] + W_0[x(2) + x(6)W^0]\} + W_0\{[x(1) + x(5)W^0] + W_0[x(3) + x(7)W^0]\}$$
$$X(1) = \{[x(0) + x(4)W^4] + W^2[x(2) + x(6)W^4]\} + W^1\{[x(1) + x(5)W^4] + W^2[x(3) + x(7)W^4)\}$$
$$X(2) = \{[x(0) + x(4)W^0] + W^4[x(2) + x(6)W^0]\} + W^2\{[x(1) + x(5)W^0] + W^4[x(3) + x(7)W^0]\}$$
$$X(3) = \{[x(0) + x(4)W^4] + W^6[x(2) + x(6)W^4]\} + W^3\{[x(1) + x(5)W^4] + W^6[x(3) + x(7)W^4]\}$$
$$X(4) = \{[x(0) + x(4)W^0] + W^0[x(2) + x(6)W^0]\} + W^4\{[x(1) + x(5)W^0] + W^0[x(3) + x(7)W^0]\}$$
$$X(5) = \{[x(0) + x(4)W^4] + W^2[x(2) + x(6)W^4]\} + W^5\{[x(1) + x(5)W^4] + W^2[x(3) + x(7)W^4]\}$$
$$X(6) = \{[x(0) + x(4)W^0] + W^4[x(2) + x(6)W^0]\} + W^6\{[x(1) + x(5)W^0] + W^4[x(3) + x(7)W^0]\}$$
$$X(7) = \{[x(0) + x(4)W^4] + W^6[x(2) + x(6)W^4]\} + W^7\{[x(1) + x(5)W^4] + W^6[x(3) + x(7)W^4]\}$$

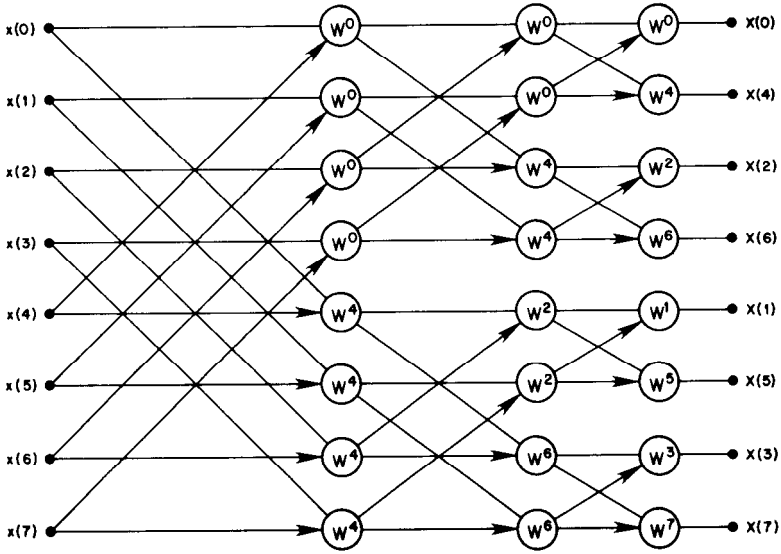

Figure 8.1 Signal flow graph representing the equations of Table 8.3.

tion scheme depicted by Fig. 8.1 can be extended for any value of N that is an integer power of 2. A C function **fft( )** that implements this scheme is given in Listing 8.3. Listings of the support functions **bitRev( ), ipow( )**, and **log2( )** are provided in App. A.

8.5 Applying the Discrete Fourier Transform

Short time-limited signals

Consider the time-limited continuous-time signal and its continuous spectrum shown in Figs. 8.2*a* and 8.2*b*. (Remember that a signal cannot be both strictly time limited and strictly band limited.) We can sample this signal to produce the time sequence shown in Fig. 8.2*c* for input to a DFT. If the input record length N is chosen to be longer than the length of the input time sequence, the entire sequence can fit within the input record as shown. As discussed in Sec. 8.2, the DFT will treat the input sequence as though it is the periodic sequence shown in Fig. 8.2*d*. This will result in a periodic discrete-frequency spectrum as shown in Fig. 8.2*e*. The actual output produced by the DFT algorithm will be the sequence of values from $m = 0$ to $m = N - 1$. Of course, there will be some aliasing due to the time-limited nature (and consequently unlimited bandwidth) of the input-signal pulse.

Periodic signals

Consider the band-limited and periodic continuous-time signal and its spectrum shown in Fig. 8.3. We can sample this signal to produce the time

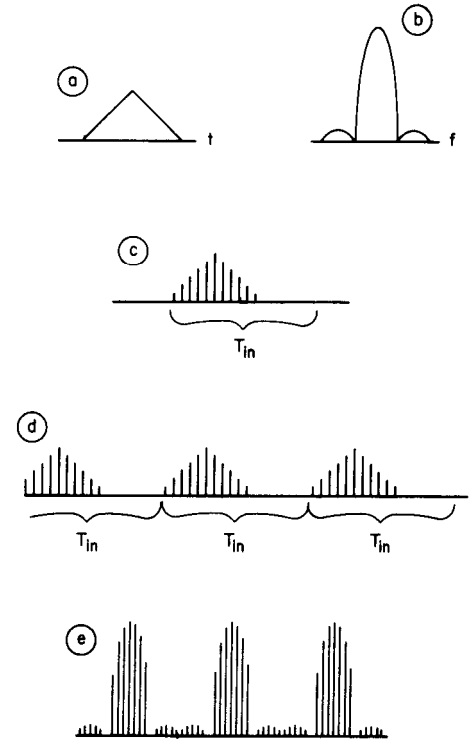

Figure 8.2 Signals and sequences for the DFT of a short time-limited signal.

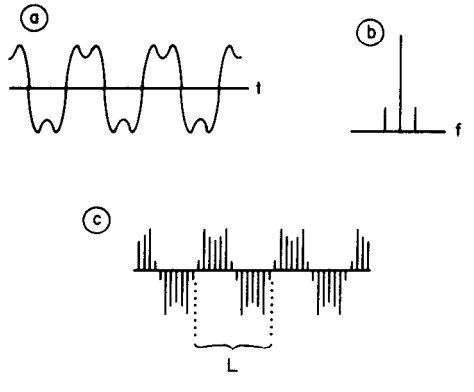

Figure 8.3 Signal and sequences for the DFT of a periodic signal. The length L of the DFT input record equals the period of the signal.

sequence shown in Fig. 8.3*c* for input to the DFT. If the input record length N of the DFT is chosen to be exactly equal to the length of 1 period of this sequence, the periodic assumption implicit in the DFT will cause the DFT to treat the single input record as though it were the complete sequence. The corresponding periodic discrete-frequency spectrum is shown in Fig. 8.3*d*. The DFT output sequence will actually consist of just 1 period that matches *exactly* the spectrum of Fig. 8.3*b*. We could not hope for (or find) a more convenient situation. Unfortunately, this realtionship exists only in an N-point DFT where the input signal is both band limited and periodic with a period of exactly N.

Long aperiodic signals

So far we have covered the use of the DFT under relatively favorable conditions that are not likely to exist in many important signal processing applications. Often the signal to be analyzed will be neither periodic nor reasonably time limited. The corresponding sequence of digitized-signal values will be longer than the DFT input record and will therefore have to be truncated to just N samples before the DFT can be applied. The periodic nature of the DFT will cause the truncated sequence of Fig. 8.4*b* to be interpreted as though it were the sequence shown in Fig. 8.4*c*. Notice that in this sequence there is a large discontinuity in the signal at the points corresponding to the ends of the input record. This will introduce additional high-frequency components into the spectrum produced by the DFT. This

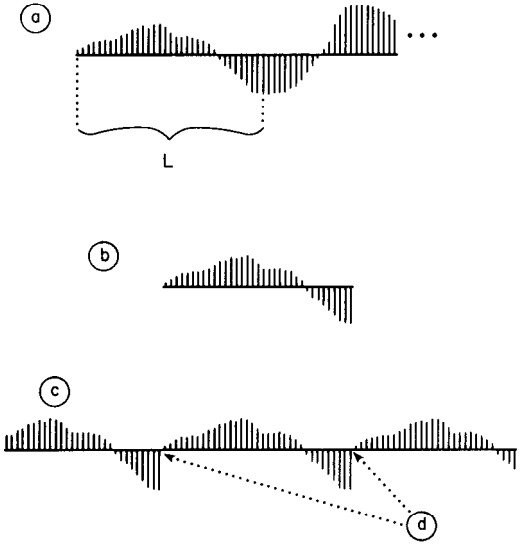

Figure 8.4 Discontinuities caused by truncating the input sequence of a DFT: (*a*) long input sequence, (*b*) truncated input sequence; (c) input sequence as interpreted by the DFT, and (*d*) resulting discontinuities.

phenomenon is called *leakage*. To reduce the leakage effects, it is a common practice to multiply the truncated input sequence by a tapering window prior to application of the DFT. A good window shape will taper off at the ends of the input record but still have a reasonably compact and narrow spectrum. This is important since multiplying the time sequence by the window will cause the corresponding frequency sequence to be convolved with the spectrum of the window. A narrow window spectrum will cause minimum smearing of the signal spectrum. Several popular windowing functions and their spectra are treated at length in Chap. 11.

Listing 8.1 dft( )

```
/**********************************/
/*                                */
/*   Listing 8.1                  */
/*                                */
/*   dft()                        */
/*                                */
/**********************************/

void dft(    struct complex x[],
             struct complex xx[],
             int N)
{
int n, m;
real sumRe, sumIm, phi;

for( m=0; m<N; m++) {
    sumRe = 0.0;
    sumIm = 0.0;
    for( n=0; n<N; n++) {
        phi = 2.0 * PI * m * n /N;
        sumRe += x[n].Re * cos(phi) + x[n].Im * sin(phi);
        sumIm += x[n].Im * cos(phi) - x[n].Re * sin(phi);
        }
    xx[m] = cmplx(sumRe, sumIm);
    }
return;
}
```

Listing 8.2 dft2( )

```
/**********************************/
/*                                */
/*   Listing 8.2                  */
/*                                */
/*   dft2()                       */
/*                                */
/**********************************/

void dft2(   struct complex x[],
             struct complex xx[],
             int N)
{
int n, m, k;
real sumRe, sumIm, phi;
static real cosVal[DFTSIZE], sinVal[DFTSIZE];
```

```
for( k=0; k<N; k++) {
    cosVal[k] = cos(2.0 * PI * k /N);
    sinVal[k] = sin(2.0 * PI * k /N);
    }

for( m=0; m<N; m++) {
    sumRe = 0.0;
    sumIm = 0.0;

    for(n=0; n<N; n++) {
        k = (m*n)%N;
        sumRe += x[n].Re * cosVal[k] + x[n].Im * sinVal[k];
        sumIm += x[n].Im * cosVal[k] - x[n].Re * sinVal[k];
        }
    xx[m] = cmplx(sumRe, sumIm);
    }
return;
}
```

Listing 8.3 fft( )

```
/***********************************/
/*                                 */
/*   Listing 8.3                   */
/*                                 */
/*   fft()                         */
/*                                 */
/***********************************/

void fft(    struct complex xIn[],
             struct complex xOut[],
             int N)
{
static struct complex x[2][DFTSIZE];
static real cc[DFTSIZE], ss[DFTSIZE];
static char inString[81];
int ping, pong, n, L, nSkip, level, nG, nB, kt, kb, nBot;

for( n=0; n<N; n++) {
    x[0][n] = xIn[n];
    cc[n] = cos(2.0 * PI * n /N);
    ss[n] = sin(2.0 * PI * n /N);
    }
```

```
pong = 0;
ping = 1;
L = log2(N);
nSkip = N;

for(level=1; level<=L; level++) {
    nSkip /= 2;
    n = 0;
    for(nG=0; nG<ipow(2,(level-1)); nG++) {
        kt = bitRev(L,2*nG);
        kb = bitRev(L,2*nG+1);

        for(nB=0; nB<nSkip; nB++) {
            nBot = n + nSkip;
            x[ping][n].Re =  x[pong][n].Re
                                + cc[kt] * x[pong][nBot].Re
                                + ss[kt] * x[pong][nBot].Im;
            x[ping][n].Im =  x[pong][n].Im
                                + cc[kt] * x[pong][nBot].Im
                                - ss[kt] * x[pong][nBot].Re;
            x[ping][nBot].Re =   x[pong][n].Re
                                + cc[kb] * x[pong][nBot].Re
                                + ss[kb] * x[pong][nBot].Im;
            x[ping][nBot].Im =   x[pong][n].Im
                                + cc[kb] * x[pong][nBot].Im
                                - ss[kb] * x[pong][nBot].Re;

            n++;
            }
        n += nSkip;
        }
    ping = !ping;
    pong = !pong;
    }
for(n=0; n<N; n++) xOut[bitRev(L,n)] = x[pong][n];
return;
}
```