Digital Signal Processing

The Fast Fourier Transform

Moslem Amiri, Václav Přenosil
Masaryk University

Relationship of FFT to DFT

- Radix-2 FFT algorithm
  - A very efficient process for performing DFTs under constraint that DFT size be an integral power of two
  - Radix-2 FFT greatly reduces the number of necessary arithmetic operations
  - The number of complex multiplications necessary for an *N*-point DFT is N^2

$$X(m) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi nm/N}$$

  - The number of complex multiplications for an *N*-point FFT is approximately $(N/2)\log_2 N$

Relationship of FFT to DFT

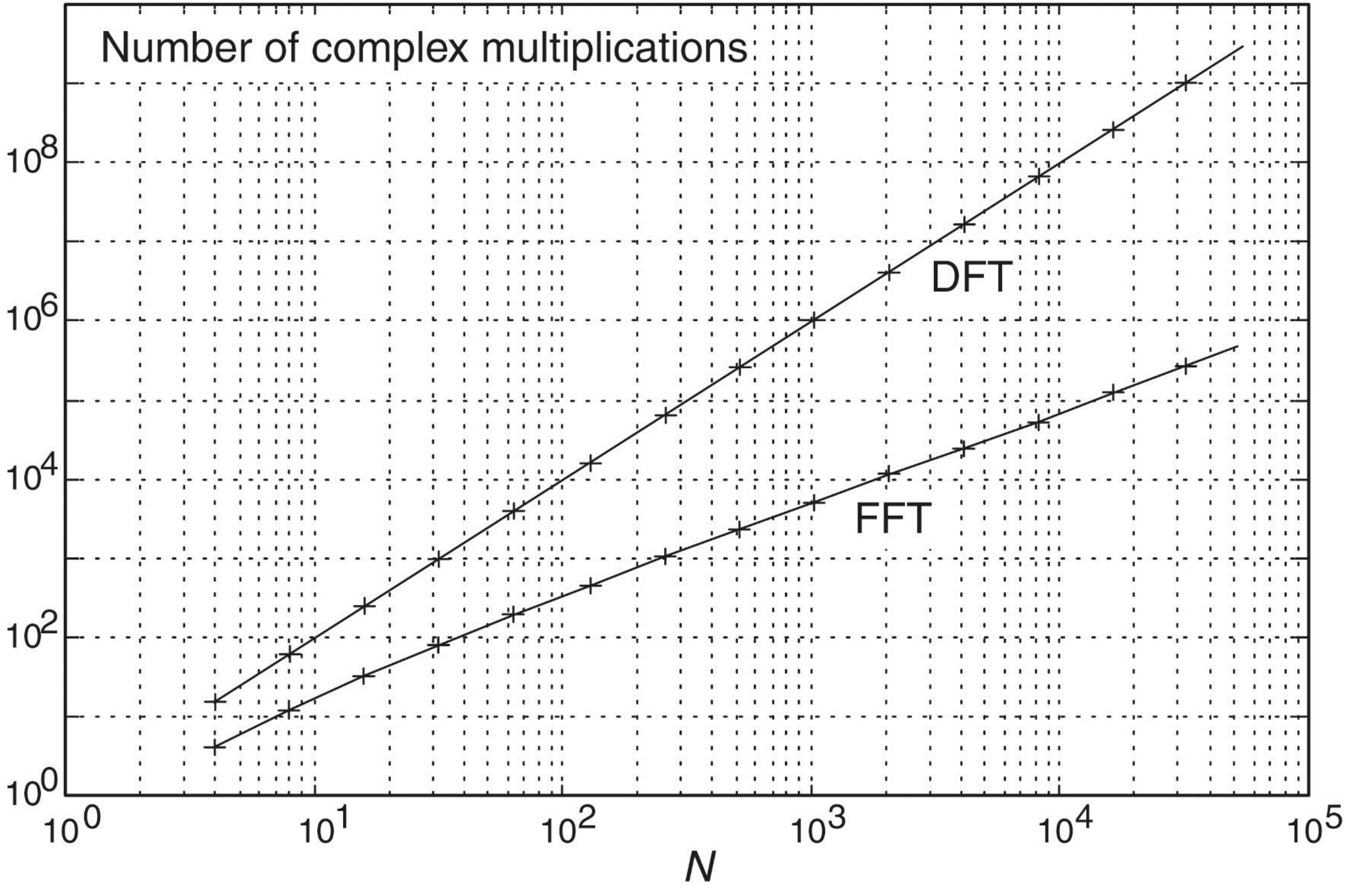

Figure 4–1 Number of complex multiplications in the DFT and the radix-2 FFT as a function of *N*.

Relationship of FFT to DFT

- FFT is not an approximation of DFT
  - It’s exactly equal to DFT
  - All of performance characteristics of DFT, output symmetry, linearity, output magnitudes, leakage, scalloping loss, etc., also describe the behavior of FFT

Hints on Using FFTs in Practice

- Sample fast enough and long enough
  - Sampling rate must be greater than twice the bandwidth of continuous A/D input signal
    - Sample at 2.5 to 4 times the signal bandwidth
    - If we don't know signal's bandwidth, we should mistrust any FFT results that have significant spectral components at frequencies near half f_s
    - Be suspicious of aliasing if there are any spectral components whose frequencies depend on f_s
    - If we suspect that aliasing is occurring or continuous signal contains broadband noise, we'll have to use an analog lowpass filter prior to A/D conversion
    - Cutoff frequency of lowpass filter must be greater than frequency band of interest but less than half f_s

Hints on Using FFTs in Practice

- Sample fast enough and long enough
  - How many samples must we collect
    - Data collection time interval must be long enough to satisfy desired FFT frequency resolution for given f_s
    - Total data collection time interval is N/f_s seconds, and N-point FFT bin-to-bin frequency resolution is f_s/N Hz
    - For example, if we need a spectral resolution of 5 Hz, then f_s/N = 5 Hz, and

$$N = \frac{f_s}{\text{desired resolution}} = \frac{f_s}{5} = 0.2\, f_s$$

    - If f_s is, say, 10 kHz, then N must be at least 2000, and we'd choose N = 2048 because this number is a power of two

Hints on Using FFTs in Practice

- Manipulating time data prior to transformation
  - If length of time-domain data sequence is not an integral power of two, we have two options
  - Discard enough data samples so that remaining sequence length is some integral power of two
    - Not recommended
  - A better approach is to append enough zero-valued samples to the end of time data sequence to match the number of points of the next largest radix-2 FFT
    - *Zero-padding* technique

Hints on Using FFTs in Practice

- Manipulating time data prior to transformation
  - We can multiply time data by a window function to alleviate leakage problem
    - But frequency resolution is degraded when windows are used
  - If appending zeros is necessary to extend a time sequence, append zeros *after* multiplying original time data sequence by a window function

Hints on Using FFTs in Practice

- Manipulating time data prior to transformation
  - Even when windowing is employed, high-level spectral components can obscure nearby low-level spectral components
    - This is especially evident when original time data has a nonzero average, i.e., it's riding on a DC bias
    - A large-amplitude DC spectral component at 0 Hz will overshadow its spectral neighbors
    - We can eliminate this problem by calculating average of time sequence and subtracting that average value from each sample in original sequence
    - The averaging and subtraction process must be performed before windowing

Hints on Using FFTs in Practice

- Enhancing FFT results
  - To detect signal energy in presence of noise (enough time-domain data is available), we can improve sensitivity of processing by averaging multiple FFTs
  - A 2*N*-point real sequence can be transformed with a single *N*-point complex radix-2 FFT to speed up our processing
  - If we need FFT of unwindowed and also windowed time-domain data, we can perform FFT of unwindowed data, and then we can perform frequency-domain windowing to reduce spectral leakage on any, or all, of FFT bin outputs

Hints on Using FFTs in Practice

- Interpreting FFT results
  - First step in interpreting FFT results is to compute absolute frequency of individual FFT bin centers
    - Like DFT, FFT bin spacing is f_s/N
    - For $m = 0, 1, 2, 3, . . ., N-1$, absolute frequency of mth bin center is mf_s/N
  - If FFT's input time samples are real, only $X(m)$ outputs from $m = 0$ to $m = N/2$ are independent
    - We need determine only absolute FFT bin frequencies for m over range of $0 \le m \le N/2$
    - If FFT input samples are complex, all N of FFT outputs are independent, and we should compute absolute FFT bin frequencies for m over range of $0 \le m \le N-1$

Hints on Using FFTs in Practice

- Interpreting FFT results
  - We can determine true amplitude of time-domain signals from their FFT spectral results
    - Radix-2 FFT outputs are complex

$$X(m) = X_{\text{real}}(m) + jX_{imag}(m)$$

    - FFT output magnitude samples

$$X_{\text{mag}}(m) = |X(m)| = \sqrt{X_{\text{real}}(m)^2 + X_{\text{imag}}(m)^2}$$

      are all inherently multiplied by factor *N*/2, when input samples are real
    - If FFT input samples are complex, scaling factor is *N*
    - So to determine correct amplitudes of time-domain sinusoidal components, divide FFT magnitudes by *N*/2 for real inputs and *N* for complex inputs

Hints on Using FFTs in Practice

- Interpreting FFT results
  - If a window function was used on original time-domain data, some of FFT input samples will be attenuated
    - This reduces the resultant FFT output magnitudes from their true unwindowed values
    - To calculate correct amplitudes of various time-domain sinusoidal components, we have to further divide FFT magnitudes by appropriate processing loss factor associated with the window function used

Hints on Using FFTs in Practice

- Interpreting FFT results
  - To determine power spectrum $X_{\mathrm{PS}}(m)$

$$X_{PS}(m) = |X(m)|^2 = X_{\mathrm{real}}(m)^2 + X_{\mathrm{imag}}(m)^2$$

$$X_{dB}(m) = 10 \cdot \log_{10}(|X(m)|^2) \ \mathrm{dB}$$

$$\text{normalized } X_{dB}(m) = 10 \cdot \log_{10}\left(\frac{|X(m)|^2}{(|X(m)|_{\max})^2}\right)$$

$$\text{normalized } X_{dB}(m) = 20 \cdot \log_{10}\left(\frac{|X(m)|}{|X(m)|_{\max}}\right)$$

  - Normalization through division by $(|X(m)|_{\max})^2$ or $|X(m)|_{\max}$ eliminates effect of any absolute FFT scale factor (N or $N/2$) or window scale factor
    - No compensation need be performed

Hints on Using FFTs in Practice

- Interpreting FFT results
  - Phase angles $X_ø(m)$

$$X_\phi(m) = \tan^{-1}\left(\frac{X_{imag}(m)}{X_{\mathrm{real}}(m)}\right)$$

    - Our calculations (or compiler) should detect occurrences of $X_{\mathrm{real}}(m) = 0$ and set corresponding $X_ø(m)$ to 90° if $X_{\mathrm{imag}}(m) > 0$, set $X_ø(m)$ to 0° if $X_{\mathrm{imag}}(m) = 0$, and set $X_ø(m)$ to −90° if $X_{\mathrm{imag}}(m) < 0$
  - FFT outputs containing significant noise components can cause large fluctuations in the computed $X_ø(m)$ phase angles
    - $X_ø(m)$ samples are meaningful when corresponding $|X(m)|$ is well above average FFT output noise level

Derivation of Radix-2 FFT Algorithm

$$X(m) = \sum_{n=0}^{N-1} x(n)\, e^{-j2\pi nm/N}$$

$$= \sum_{n=0}^{(N/2)-1} x(2n)\, e^{-j2\pi(2n)m/N} + \sum_{n=0}^{(N/2)-1} x(2n+1)\, e^{-j2\pi(2n+1)m/N}$$

$$\xrightarrow{W_N = e^{-j2\pi/N}} = \sum_{n=0}^{(N/2)-1} x(2n) W_N^{2nm} + W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1) W_N^{2nm}$$

$$\xrightarrow[=e^{-j2\pi/(N/2)}=W_{N/2}]{W_N^2 = e^{-j2\pi 2/(N)}} = \sum_{n=0}^{(N/2)-1} x(2n) W_{N/2}^{nm} + W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1) W_{N/2}^{nm}$$

- where *m* is in range 0 to *N*/2−1
  - Index *m* has that reduced range because each of the two *N*/2-point DFTs on the right side are periodic in *m* with period *N*/2

Derivation of Radix-2 FFT Algorithm

$$\xrightarrow{W_{N/2}=e^{-j2\pi/(N/2)}} X(m) = \sum_{n=0}^{(N/2)-1} x(2n)W_{N/2}^{nm} + W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1)W_{N/2}^{nm}$$

- We have two *N*/2 summations whose results can be combined to give the first *N*/2 samples of an *N*-point DFT
- Benefits of breaking *N*-point DFT into two parts
  - Reduction of number crunching because *W* terms in the two summations are identical
  - Also the upper half of DFT outputs is easy to calculate

Derivation of Radix-2 FFT Algorithm

$$\xrightarrow{W_{N/2}=e^{-j2\pi/(N/2)}} X(m) = \sum_{n=0}^{(N/2)-1} x(2n)W_{N/2}^{nm} + W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1)W_{N/2}^{nm}$$

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

$$W_{N/2}^{n(m+N/2)} = W_{N/2}^{nm}W_{N/2}^{nN/2} = W_{N/2}^{nm}(e^{-j2\pi n2N/2N}) = W_{N/2}^{nm}(1) = W_{N/2}^{nm}$$

$$\xrightarrow{\text{twiddlefactor}} W_N^{(m+N/2)} = W_N^m W_N^{N/2} = W_N^m(e^{-j2\pi N/2N}) = W_N^m(-1) = -W_N^m$$

$$X(m+N/2) = \sum_{n=0}^{(N/2)-1} x(2n)W_{N/2}^{nm} - W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1)W_{N/2}^{nm}$$

- We just change sign of twiddle factor and use results of the two summations from $X(m)$ to get $X(m+N/2)$
- m goes from 0 to $(N/2)-1$
- To compute an N-point DFT, we actually perform two $N/2$-point DFTs—one $N/2$-point DFT on even-indexed and one $N/2$-point DFT on odd-indexed $x(n)$ samples

Derivation of Radix-2 FFT Algorithm

$$X(m) = \sum_{n=0}^{(N/2)-1} x(2n) W_{N/2}^{nm}$$

$$+ W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1) W_{N/2}^{nm}$$

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

$$- W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1) W_{N/2}^{nm}$$

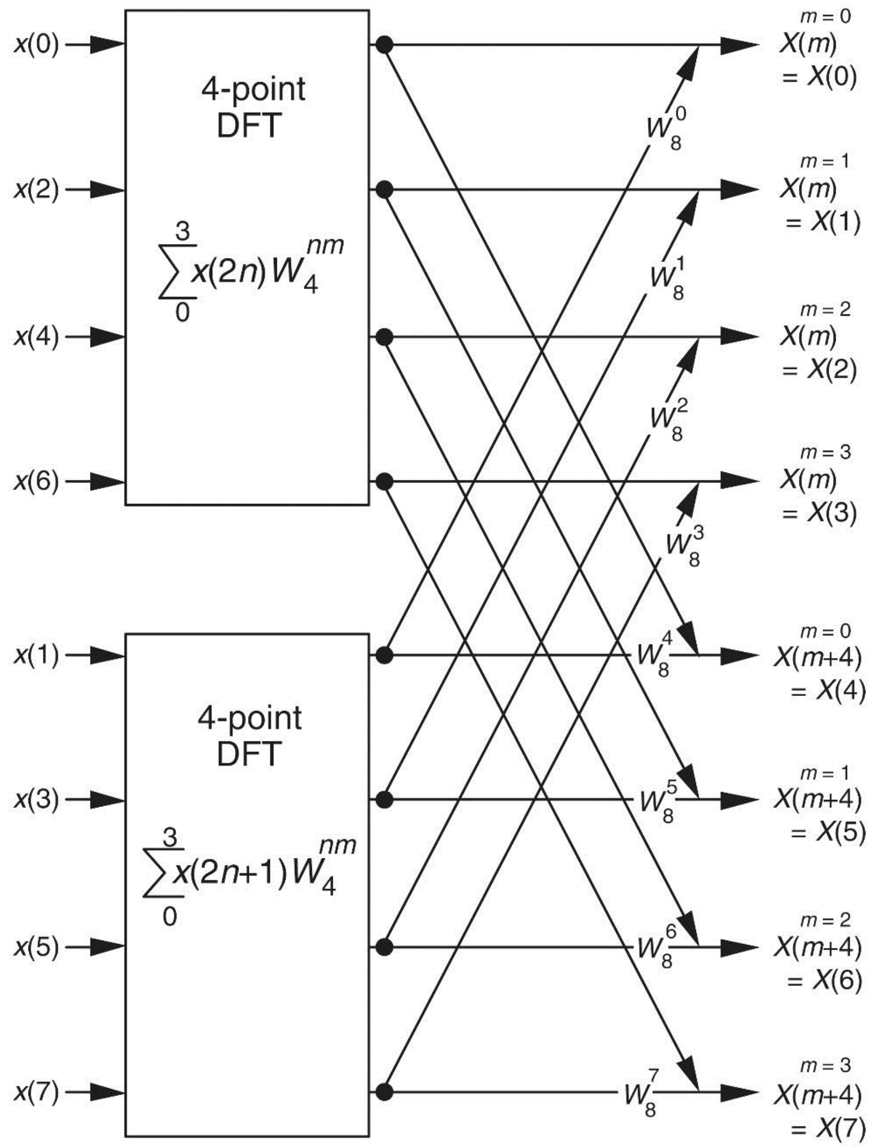

Figure 4–2 FFT implementation of an 8-point DFT using two 4-point DFTs.

Derivation of Radix-2 FFT Algorithm

$$X(m) = \sum_{n=0}^{(N/2)-1} x(2n) W_{N/2}^{nm} + W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1) W_{N/2}^{nm}$$

$$X(m+N/2) = \sum_{n=0}^{(N/2)-1} x(2n) W_{N/2}^{nm} - W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1) W_{N/2}^{nm}$$

- Twidle factors
  - Because $-e^{-j2\pi m/N} = e^{-j2\pi(m+N/2)/N}$, negative *W* twiddle factors are implemented with positive *W* twiddle factors that follow the lower DFT in Fig. 4-2

Derivation of Radix-2 FFT Algorithm

$$X(m) = \sum_{n=0}^{(N/2)-1} x(2n) W_{N/2}^{nm} + W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1) W_{N/2}^{nm}$$

$$X(m+N/2) = \sum_{n=0}^{(N/2)-1} x(2n) W_{N/2}^{nm} - W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1) W_{N/2}^{nm}$$

$$\xrightarrow{\text{simplification}} X(m) = A(m) + W_N^m B(m)$$

$$\xrightarrow{\text{simplification}} X(m+N/2) = A(m) - W_N^m B(m)$$

$$A(m) = \sum_{n=0}^{(N/2)-1} x(2n) W_{N/2}^{nm}$$

$$= \sum_{n=0}^{(N/4)-1} x(4n) W_{N/2}^{2nm} + \sum_{n=0}^{(N/4)-1} x(4n+2) W_{N/2}^{(2n+1)m}$$

$$\xrightarrow{W_{N/2}^{2nm} = W_{N/4}^{nm}} A(m) = \sum_{n=0}^{(N/4)-1} x(4n) W_{N/4}^{nm} + W_{N/2}^m \sum_{n=0}^{(N/4)-1} x(4n+2) W_{N/4}^{nm}$$

$$B(m) = \sum_{n=0}^{(N/4)-1} x(4n+1) W_{N/4}^{nm} + W_{N/2}^m \sum_{n=0}^{(N/4)-1} x(4n+3) W_{N/4}^{nm}$$

Derivation of Radix-2 FFT Algorithm

$$X(m) = A(m) + W_N^m B(m)$$

$$X(m + N/2) = A(m) - W_N^m B(m)$$

$$A(m) = \sum_{n=0}^{(N/4)-1} x(4n) W_{N/4}^{nm} + W_{N/2}^m \sum_{n=0}^{(N/4)-1} x(4n+2) W_{N/4}^{nm}$$

$$B(m) = \sum_{n=0}^{(N/4)-1} x(4n+1) W_{N/4}^{nm} + W_{N/2}^m \sum_{n=0}^{(N/4)-1} x(4n+3) W_{N/4}^{nm}$$

Twiddle factor $W_{N/2}^m$ for *N* = 8, ranges from W_4^0 to W_4^3 because the *m* index, for *A*(*m*) and *B*(*m*), goes from 0 to 3

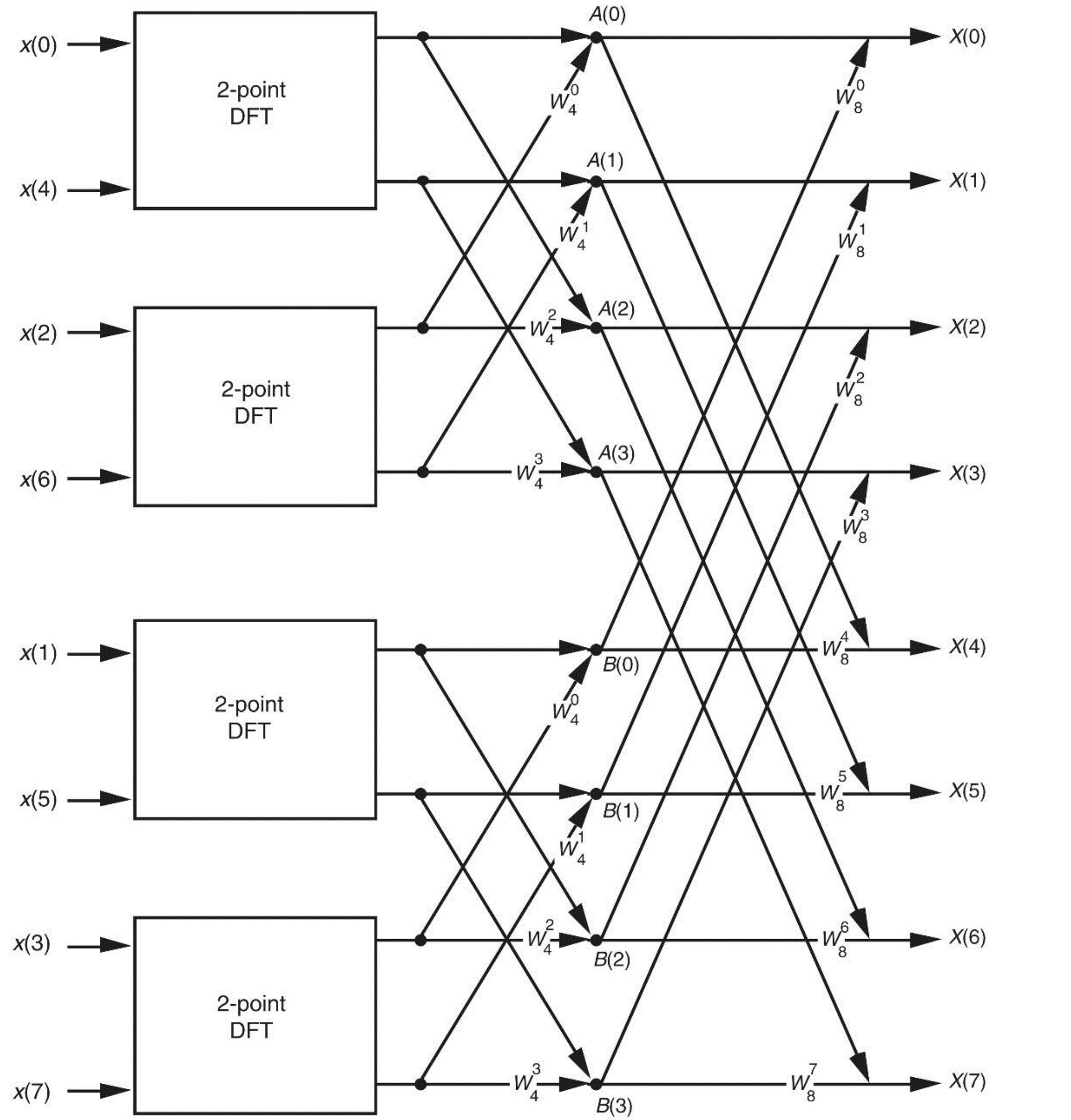

Figure 4-3 FFT implementation of an 8-point DFT as two 4-point DFTs and four 2-point DFTs.

Derivation of Radix-2 FFT Algorithm

- Fig. 4-3
  - For any *N*-point DFT, we break each of *N*/2-point DFTs into two *N*/4-point DFTs to further reduce the number of sine and cosine multiplications
  - Eventually, we arrive at an array of 2-point DFTs where no further computational savings could be realized
    - The 2-point DFT functions cannot be partitioned into smaller parts
    - Butterfly of a single 2-point DFT is shown in Fig. 4-4

Derivation of Radix-2 FFT Algorithm

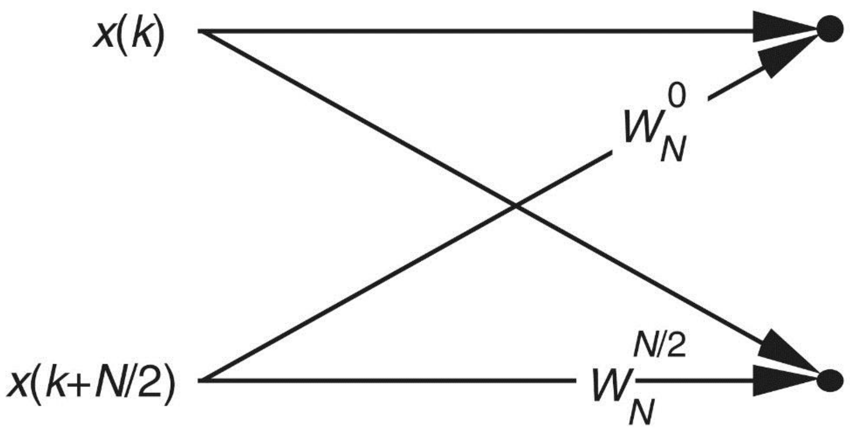

Figure 4–4 Single 2-point DFT butterfly.

- The 2-point DFT blocks in Fig. 4-3 are replaced by butterfly in Fig. 4-4 to give a full 8-point FFT implementation of DFT as shown in Fig. 4-5

$$W_N^0 = e^{-j2\pi 0/N} = 1$$

$$W_N^{N/2} = e^{-j2\pi N/2N} = e^{-j\pi} = -1$$

Derivation of Radix-2 FFT Algorithm

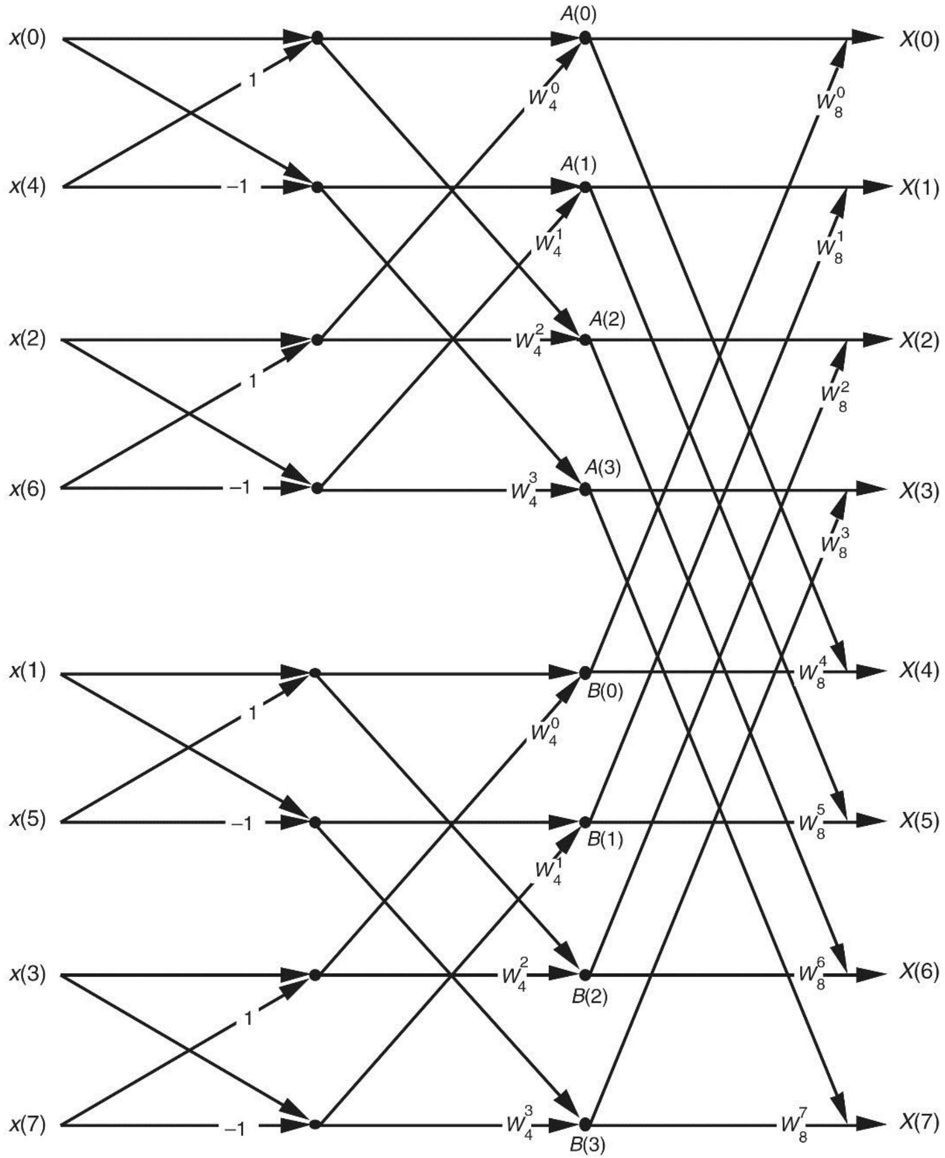

Figure 4–5 Full decimation-in-time FFT implementation of an 8-point DFT.

FFT Input/Output Data Index Bit Reversal

- Decimation-in-time FFT implementation
  - Was the title of Fig. 4-5
  - *Decimation-in-time* phrase refers to how we broke DFT input samples into odd and even parts
  - This time decimation leads to scrambled order of input data's index *n* in Fig. 4-5
  - Shuffling of input data is known as *bit reversal*
    - Because scrambled order of input data index can be obtained by reversing bits of binary representation of normal input data index order

FFT Input/Output Data Index Bit Reversal

- Input index bit reversal for an 8-point FFT

Normal order of index *n*	**Binary bits of index *n***	**Reversed bits of index *n***	**Bit-reversed order of index *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

Radix-2 FFT Butterfly Structures

- Twiddle factors in Fig. 4-5
  - To simplify signal flows, replace twiddle factors with their equivalent values referenced to W_N^m where *N* = 8
    - We show just exponents *m* of W_N^m, to get FFT structure shown in Fig. 4-8
- Fig. 4-8
  - W_4^1 from Fig. 4-5 → W_8^2
  - W_4^2 from Fig. 4-5 → W_8^4
  - …
  - 1s and −1s in the first stage of Fig. 4-5 are replaced by 0s and 4s, respectively

Radix-2 FFT Butterfly Structures

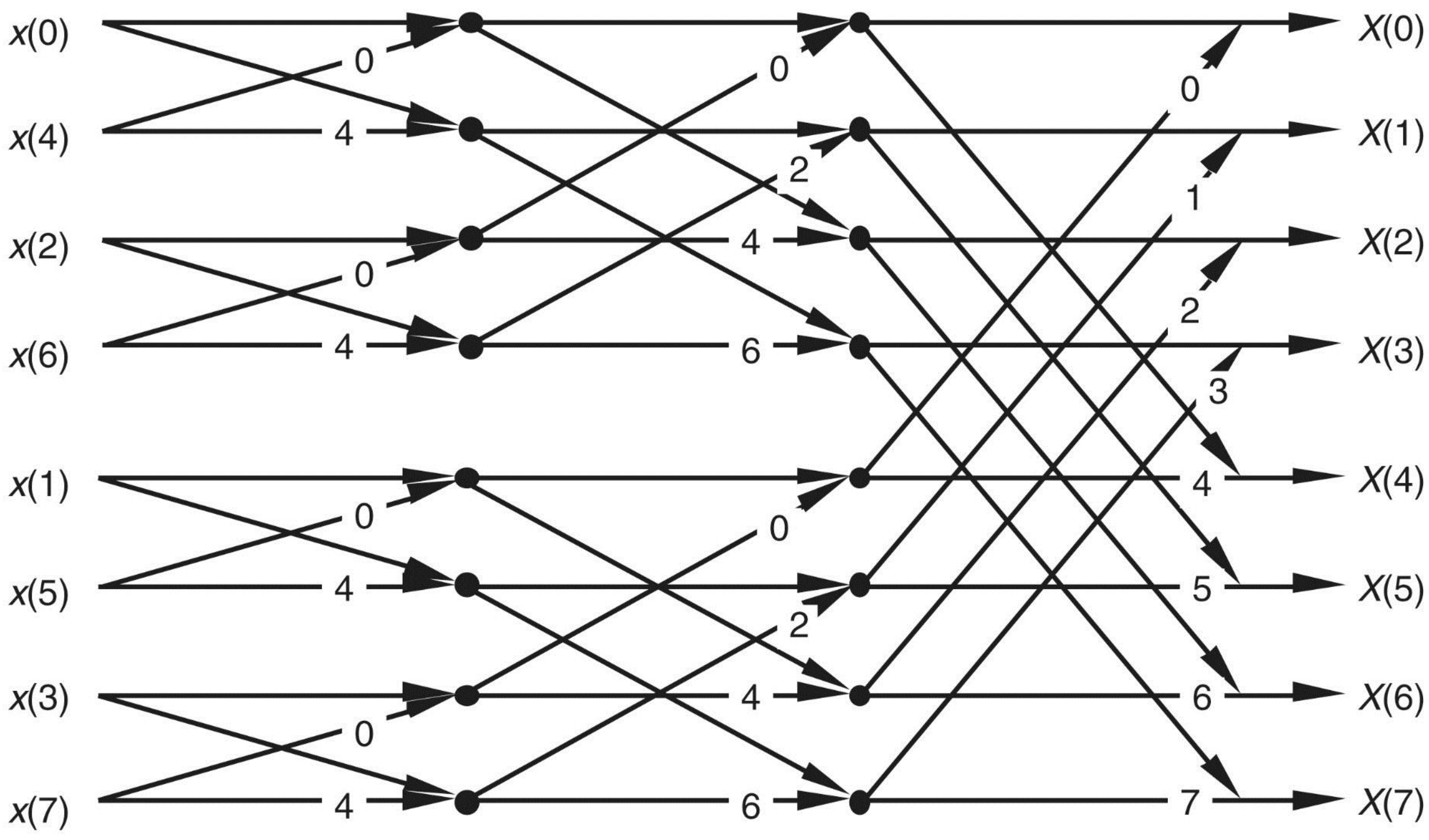

Figure 4–8 Eight-point decimation-in-time FFT with bit-reversed inputs.

Radix-2 FFT Butterfly Structures

input data is in its normal order and output data indices are bit-reversed → a bit-reversal operation needs to be performed at output of FFT to unscramble frequency-domain results

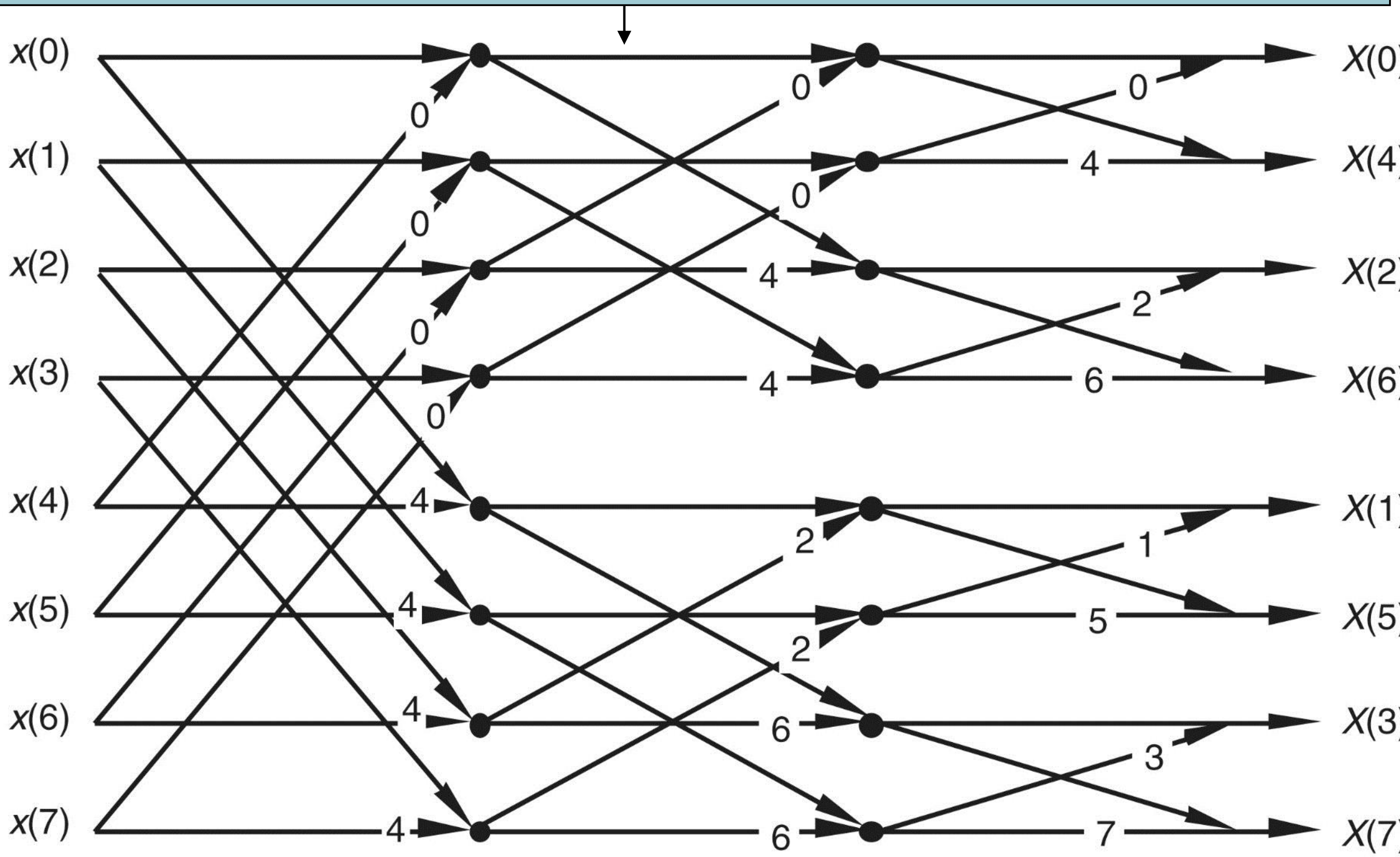

Figure 4–9 Eight-point decimation-in-time FFT with bit-reversed outputs.

Radix-2 FFT Butterfly Structures

an FFT signal-flow structure that avoids bit-reversal problem altogether

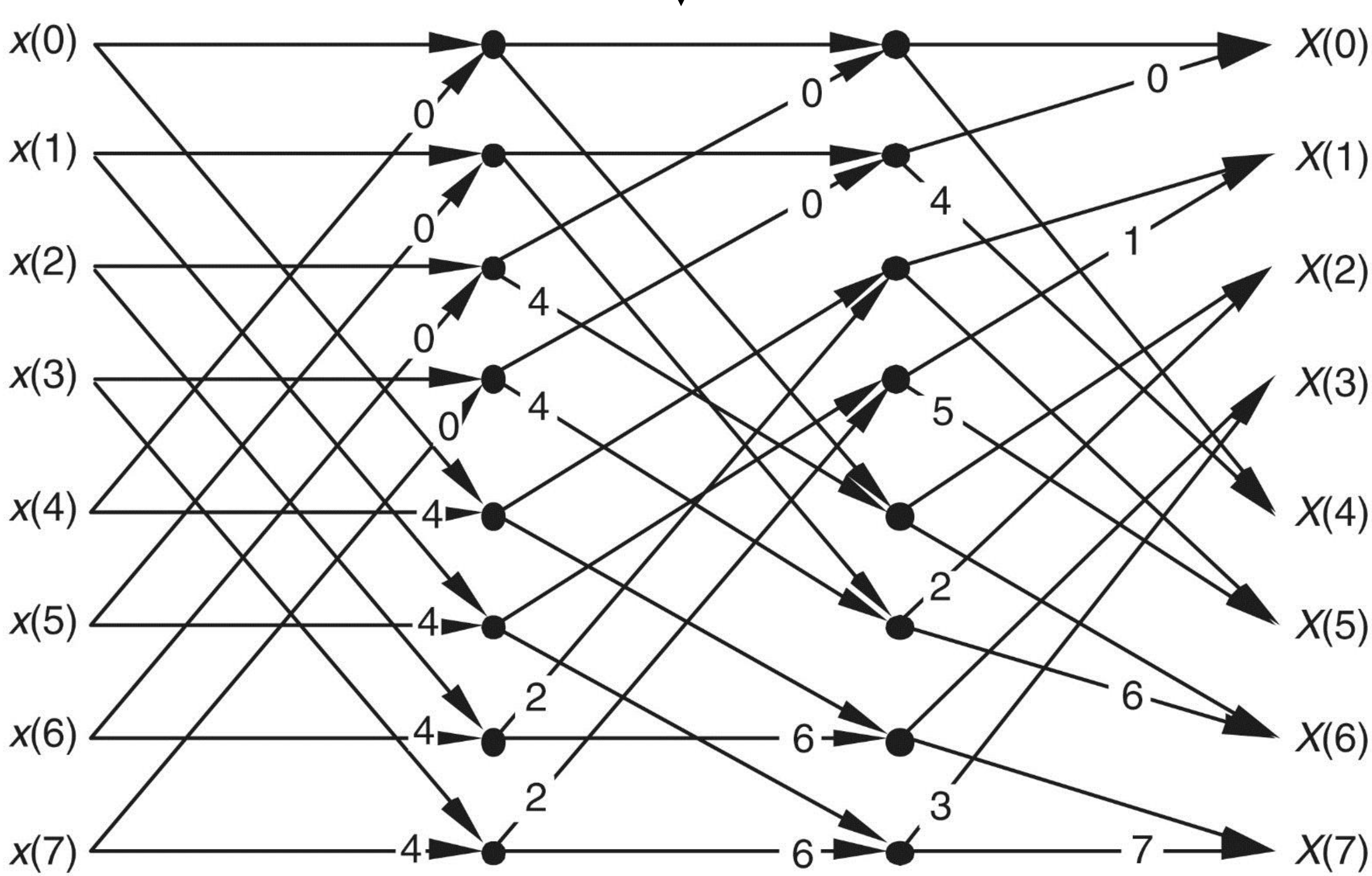

Figure 4–10 Eight-point decimation-in-time FFT with inputs and outputs in normal order.

Radix-2 FFT Butterfly Structures

- Bit reversal
  - A few years ago, hardware implementations of FFT spent most of their time performing multiplications
    - Bit-reversal process necessary to access data in memory wasn't a significant portion of overall FFT computational problem
  - Now that high-speed multiplier/accumulator integrated circuits can multiply two numbers in a single clock cycle, FFT data multiplexing and memory addressing are more important
    - Led to development of efficient algorithms to perform bit reversal

Radix-2 FFT Butterfly Structures

- Decimation-in-frequency algorithm
  - Decimation-in-time or -frequency is determined by whether the DFT inputs or outputs are partitioned (into odd and even) when deriving a particular FFT butterfly structure from the DFT equations
  - Decimation-in-frequency butterfly structures (analogous to structures in Figs. 4-8 through 4-10) are illustrated in Figs. 4-11 through 4-13
    - An equivalent decimation-in-frequency FFT structure exists for each decimation-in-time FFT structure
    - The number of necessary multiplications is the same for both structures

Radix-2 FFT Butterfly Structures

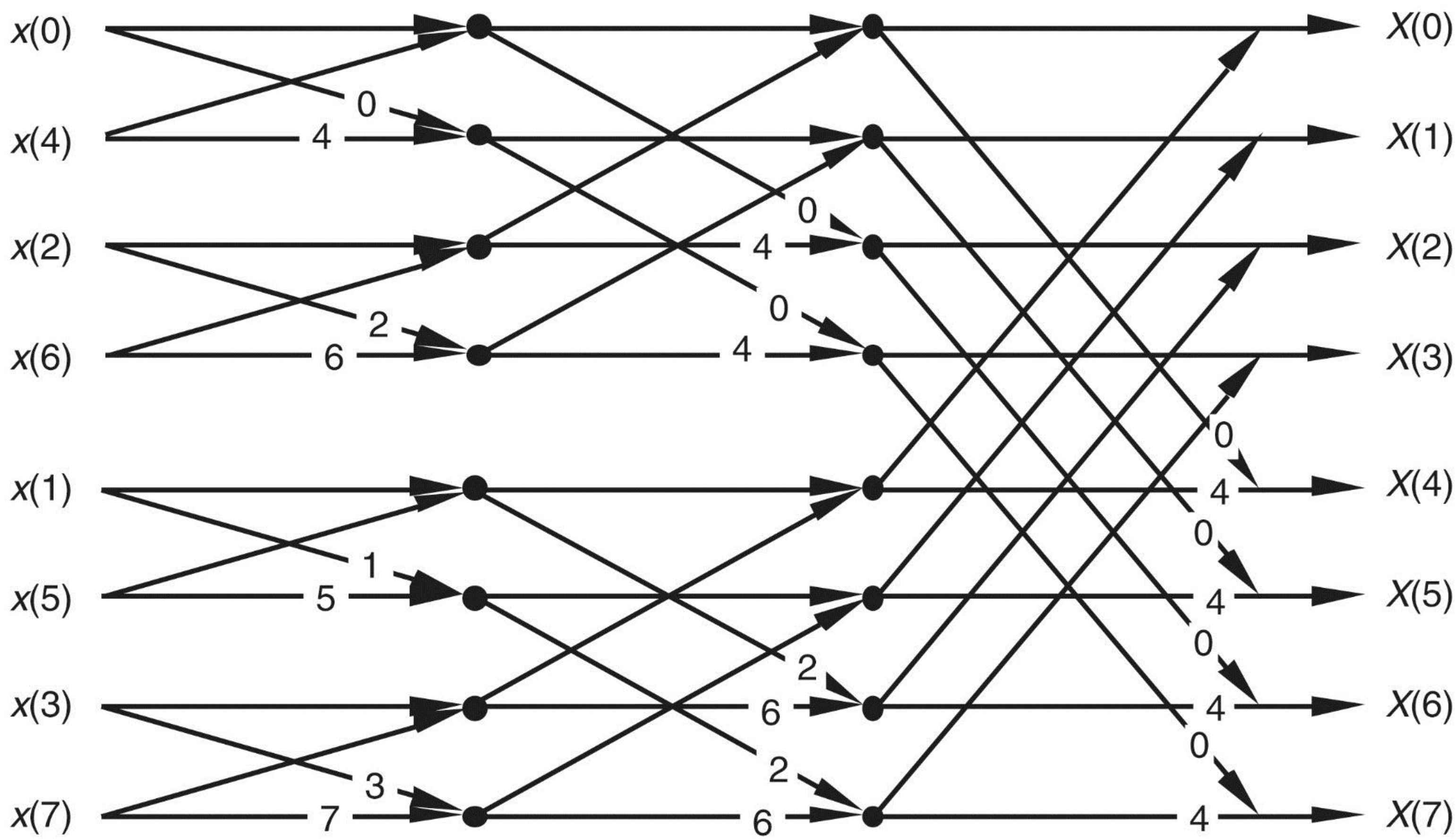

Figure 4–11 Eight-point decimation-in-frequency FFT with bit-reversed inputs.

Radix-2 FFT Butterfly Structures

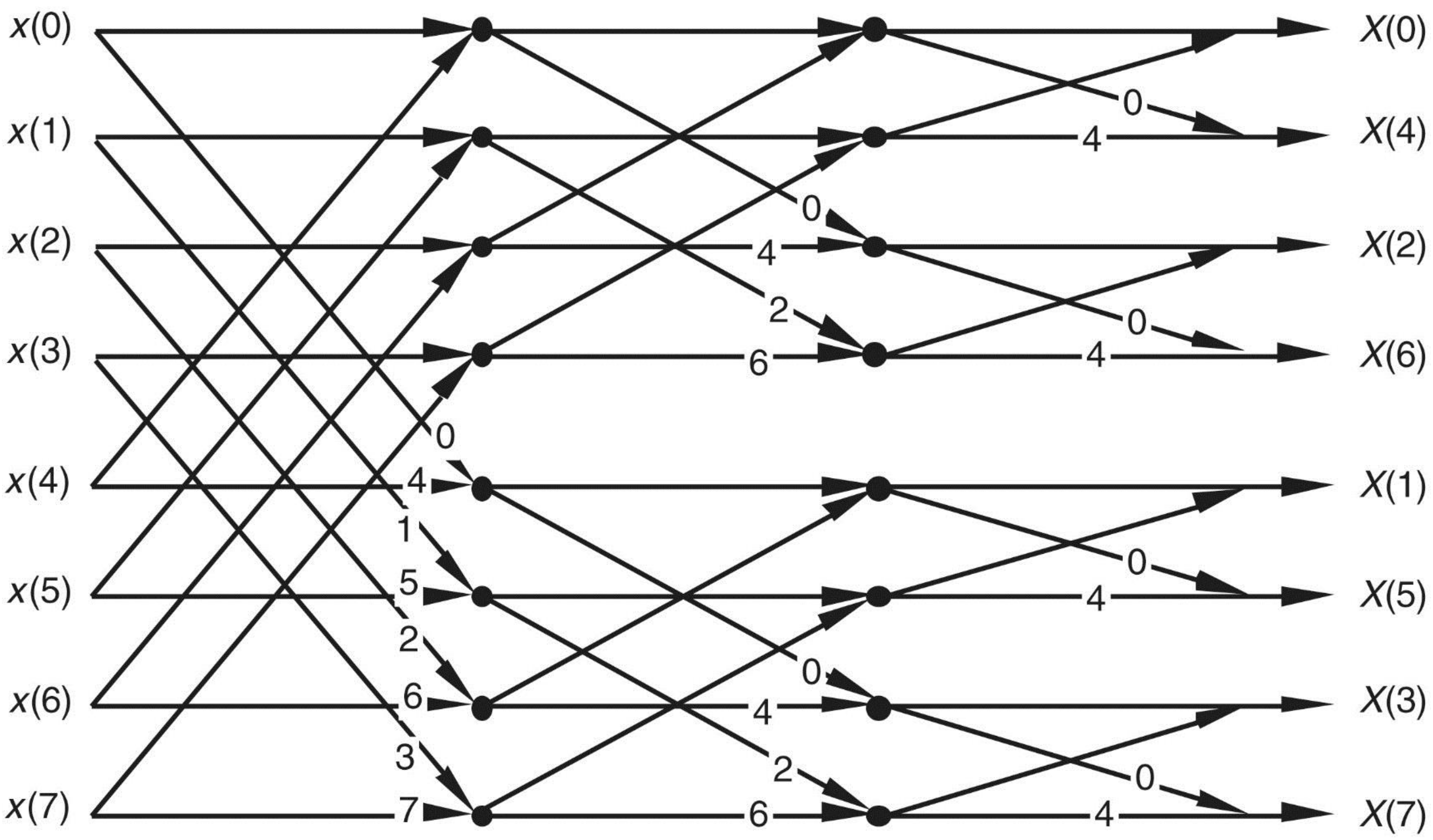

Figure 4–12 Eight-point decimation-in-frequency FFT with bit-reversed outputs.

Radix-2 FFT Butterfly Structures

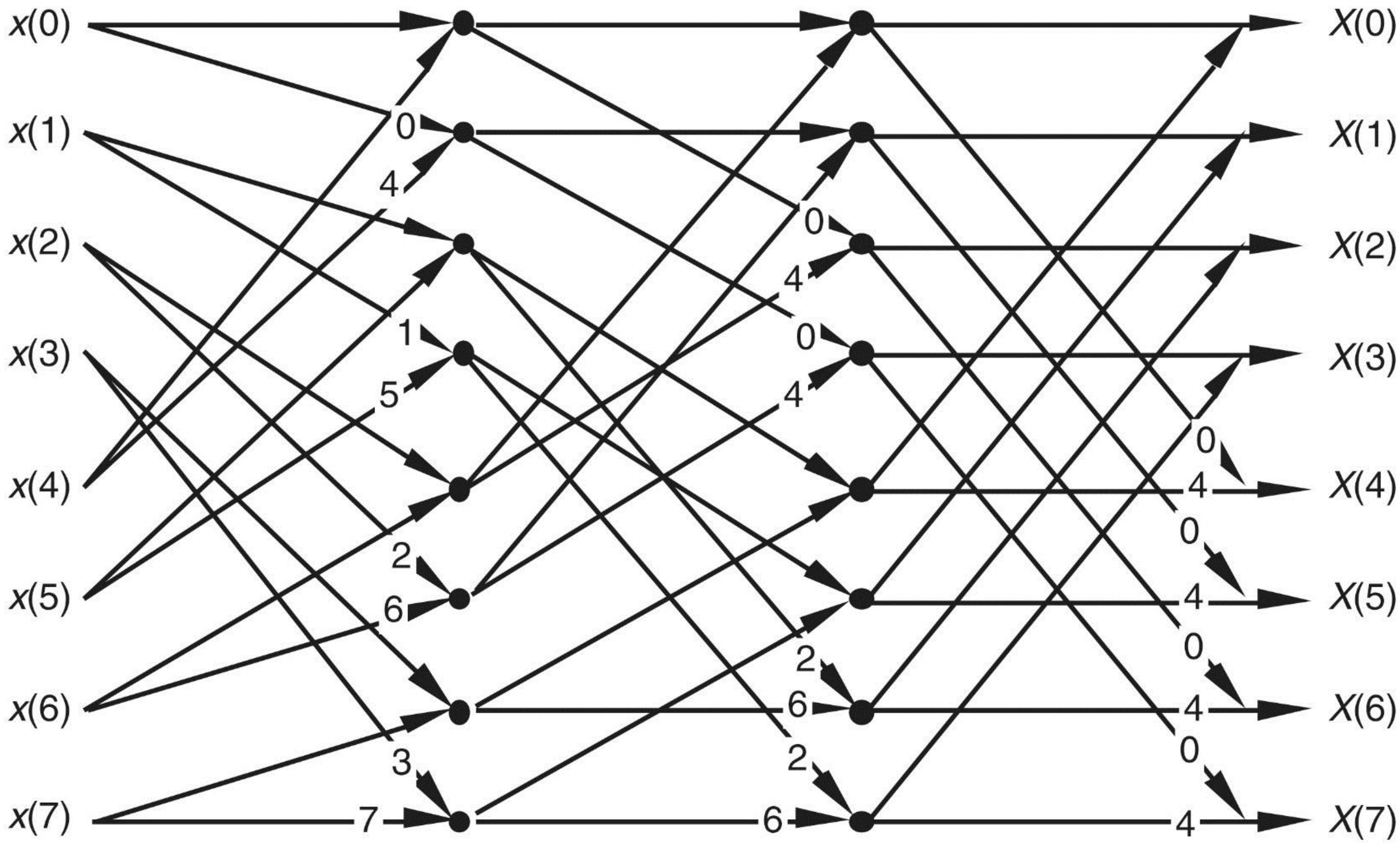

Figure 4–13 Eight-point decimation-in-frequency FFT with inputs and outputs in normal order.

Alternate Single-Butterfly Structures

- Butterfly structures
  - FFT butterfly structures are direct result of derivations of decimation-in-time and decimation-in-frequency algorithms
    - Twiddle factors always take general forms shown in Fig. 4-14(a)

Alternate Single-Butterfly Structures

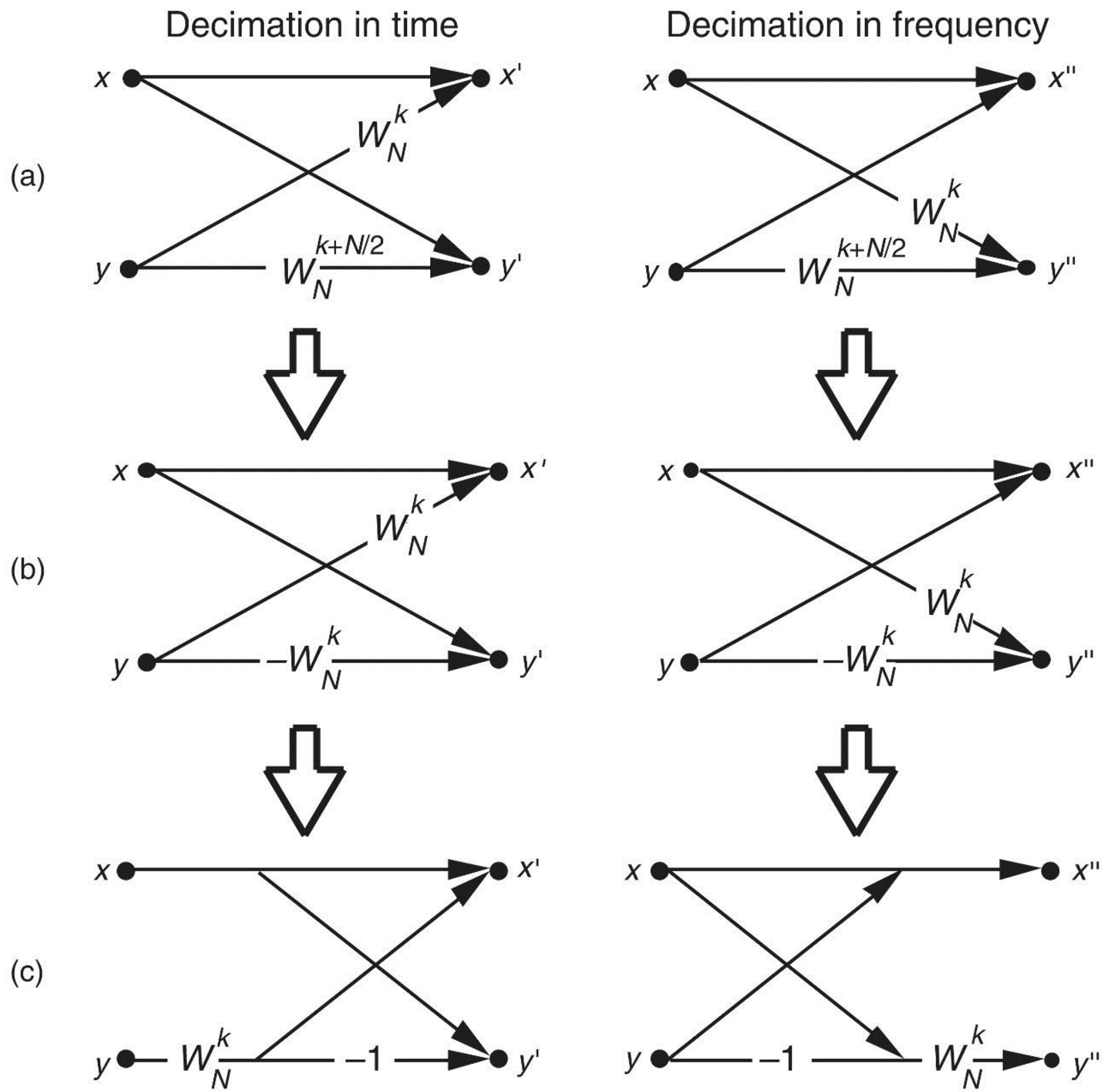

Figure 4–14 Decimation-in-time and decimation-in-frequency butterfly structures: (a) original form; (b) simplified form; (c) optimized form.

Alternate Single-Butterfly Structures

- Fig. 4-14
  - To implement decimation-in-time butterfly of (a), we have to perform two complex multiplications and two complex additions

$$x' = x + W_N^k y$$

$$y' = x + W_N^{k+N/2} y$$

$$\xrightarrow{\text{simplification}} W_N^{k+N/2} = W_N^k W_N^{N/2} = W_N^k (e^{-j2\pi N/2N}) = W_N^k(-1) = -W_N^k$$

  - So we replace $W_N^{k+N/2}$ in (a) with $-W_N^k$ to give us simplified butterflies in (b)
  - Because twiddle factors in (b) differ only by their signs, the optimized butterflies in (c) can be used

Alternate Single-Butterfly Structures

- Optimized butterflies in 4-14(c)
  - Require two complex additions but only one complex multiplication, thus reducing computational workload
  - Because there are $(N/2)\log_2 N$ butterflies in an N-point FFT, the number of complex multiplications performed by an FFT is $(N/2)\log_2 N$
  - An algorithm is decimation-in-time if the twiddle factor precedes the −1 in optimized butterflies
  - An algorithm is decimation-in-frequency if the twiddle factor follows the −1 in optimized butterflies

Alternate Single-Butterfly Structures

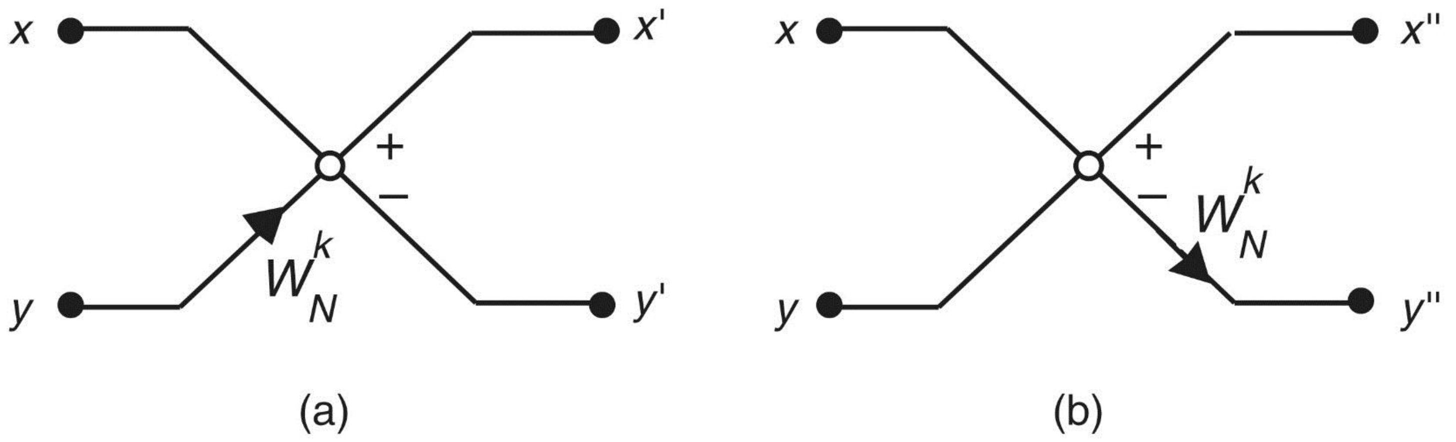

Figure 4–15 Alternate FFT butterfly notation: (a) decimation in time; (b) decimation in frequency.

Alternate Single-Butterfly Structures

- In-place FFT algorithms
  - An in-place algorithm is depicted in Fig. 4-5
  - Output of a butterfly operation can be stored in the same hardware memory locations that previously held butterfly's input data
    - No intermediate storage is necessary
  - For an *N*-point FFT, only 2*N* memory locations are needed
    - The 2 comes from fact that each butterfly node represents a data value that has both a real and an imaginary part
  - Data routing and memory addressing are rather complicated

Alternate Single-Butterfly Structures

- Double-memory FFT algorithms
  - A double-memory FFT structure is depicted in Fig. 4-10
  - Intermediate storage is necessary because we no longer have standard butterflies, and $4N$ memory locations are needed
  - Data routing and memory address control are much simpler in double-memory FFT structures than in-place technique