1. Metropolis Algorithm for Monte Carlo
2. Simplex Method for Linear Programming
3. Conjugate Gradients & Krylov Subspace Iteration Method for Solving Linear System
4. Matrix Factorizations (LU, Cholesky) for Solving Linear System and Least Squares
5. The Fortran Optimizing Compiler
6. QR/QZ/SVD Algos for Eigenvalue Computation
7. Quick Sort Algorithm
8. **Fast Fourier Transform (FFT)**
9. Fast Multipole Method
10. Integer Relation Detection

Fast Fourier Transforms

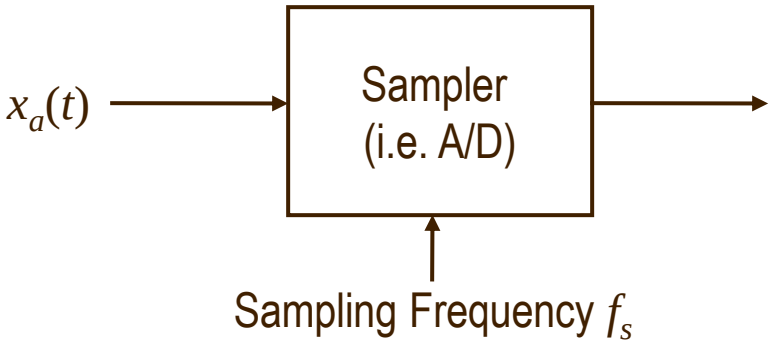
Outline

- Discrete Fourier Transform
- Fast Fourier Transform
  - Radix-2 Decimation in Time
  - Radix-2 Decimation in Frequency
  - Radix-4 Algorithms
- FFT Hardware Implementations for FPGAs and ASICs
  - Parallel FFT
  - Serial FFT (In-Place FFT)
  - Semi-Parallel FFT
  - Pipeline FFT

Discrete Fourier Transforms

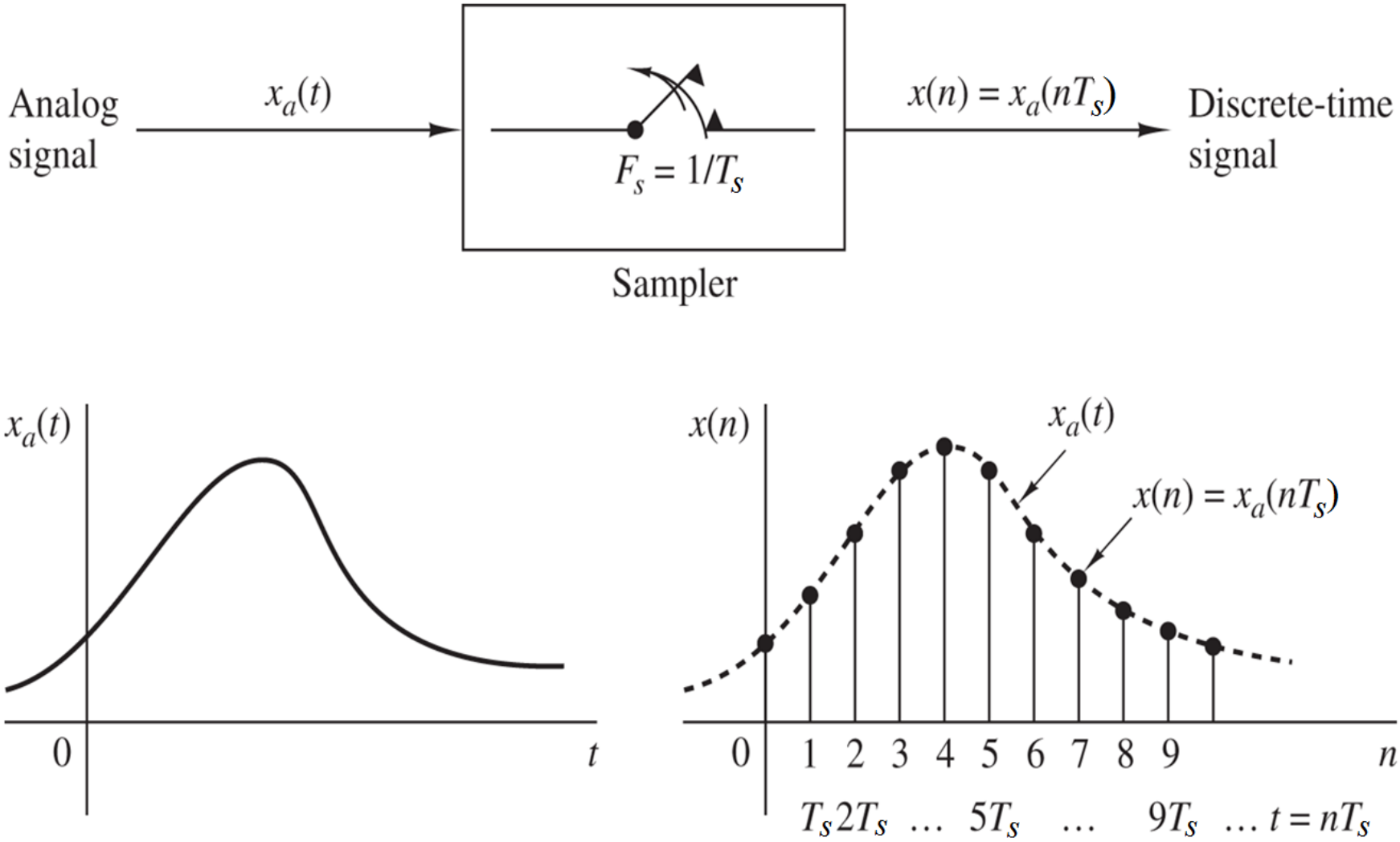
Analog to Digital Conversion

- Consider an analog signal
  - $x_a(t) = A.\cos(\omega.t) = A.\cos(2\pi.f.t)$
  - A = amplitude
  - $\omega = 2\pi.f$ = analog radian frequency (radians/sec)
  - f = analog frequency (cycles/sec = Hz)
  - $T = 1/f$ = Time Period of the Signal
- Perform sampling of analog signal at a rate of f_s (samples/sec) = sampling frequency or sampling rate
  - Sampling Period = $T_s = 1 / f_s$



(for convenience let us drop the T_s in the notation)

Analog to Digital Conversion

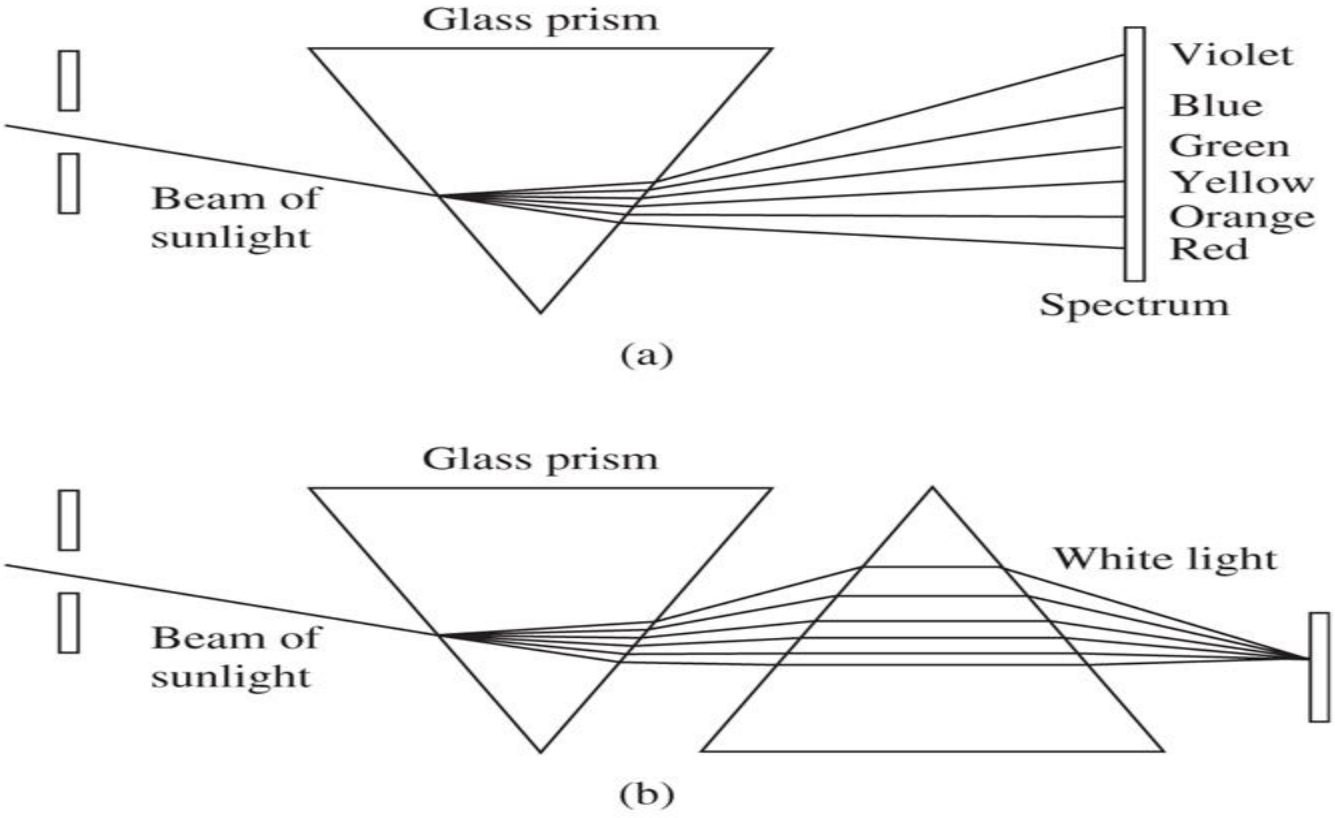


Periodic sampling of an analog signal.

Analog to Digital Conversion

- $x_a(n.T_s) = x(n) = A.\cos(2\pi.f.n.T_s) = A.\cos(2\pi.n.f/f_s)$
  - Define digital frequency $f = f/fs$
  - Define digital radian frequency $\omega = 2\pi.f$
  - $x(n) = A.\cos(2\pi.fn) = A.\cos(\omega.n)$
- A/D conversion also performs amplitude quantization converting voltage levels to bits
- To prevent aliasing $f_s >= 2f_{max}$
  - Sampling frequency f_s must be higher than twice the largest analog frequency component f_{max} (which is f in this example)
  - $2f_{max}$ is called the **Nyquist Rate**. The sampling rate must be greater than or equal to Nyquist Rate to avoid aliasing.
- The range of f: - ½ < f < ½ and the range of ω: $-\pi < \omega < \pi$

Fourier Analysis of Signals



(a) Analysis and (b) synthesis of the white light (sunlight) using glass prisms.

- Fourier analysis allows us to decompose a signal in the time domain and analyze/process the signal in the frequency domain

Discrete Fourier Transforms (DFT)

- In 1807, Jean Baptiste Joseph Fourier Introduced the Conceptual framework of Fourier Transforms.
- It allows a sampled or discrete signal that is periodic to be transformed from the time domain to the frequency domain
- It establishes the Correlation between the time domain signal and N cosine and N sine waves

Discrete Fourier Transforms (DFT)

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

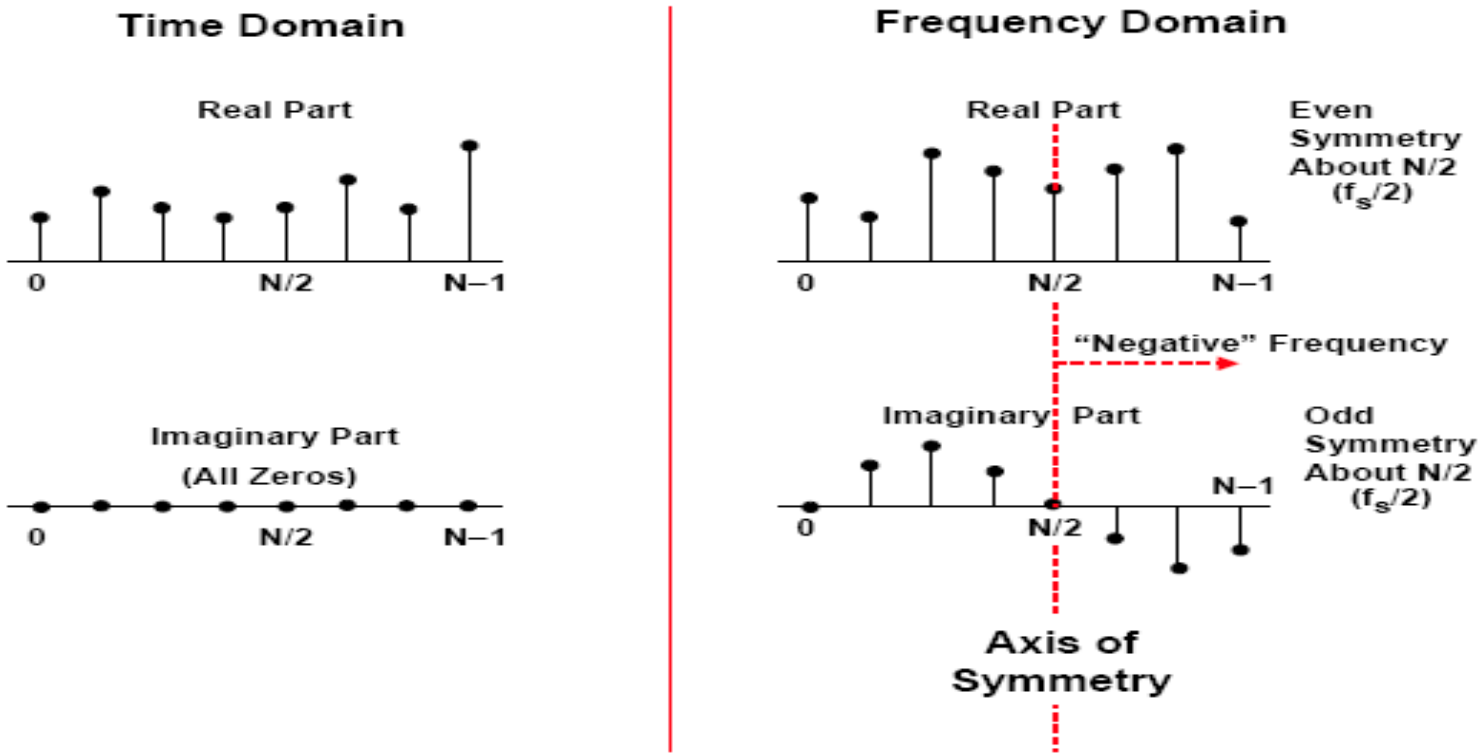
$$W_N = e^{-j2\pi/N}$$

$X(k)$ = DFT of Signal "$x(n)$"

N = Number of Sample Points

$x(n)$ = Discrete-Time Signal

W_N = Twiddle Factor



Discrete Fourier Transforms (DFT)

- DFT has the following salient features:

(i) Spectrum of discrete-time periodic signal with finite duration L can be exactly recovered from its N samples taken from 0 to 2π such that frequency ω = 2 π k/N for k = 0 to N-1 (if N >= L)

(ii) N-point DFT can be written as

$$X(\omega = 2\pi k / N) = X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi nk/N} \qquad k = 0,1,\ldots,N-1$$

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \qquad k = 0,1,\ldots,N-1 \qquad W_N = e^{-j2\pi/N}$$

Discrete Fourier Transforms (DFT)

- Forward (Direct) Discrete Fourier Transform (DFT) of a discrete-time signal $x[n]$ with finite extent $n \in [0, N\text{-}1]$

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-j\frac{2\pi}{N}nk} \quad \text{for } k = 0, 1, \ldots, N\text{-}1$$

Two-Point DFT

$$X[0] = x[0] + x[1]$$
$$X[1] = x[0] - x[1]$$

$X[k]$ periodic with period N due to exponential.

Note that in this case the points are spaced $2\pi/N$;

thus the resolution of the samples of the frequency spectrum is $2\pi/N$.

Also assume that $x[n]$ periodic with period N.

- Twiddle Factor $\quad W_N = e^{-j\frac{2\pi}{N}} \Rightarrow X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk}$

Discrete Fourier Transforms (Contd…)

Let $W_N = e^{-j\frac{2\pi}{N}} \Rightarrow N^{th}$ root of unity $(W_N^N = 1)$ since $W_N^N = e^{-j2\pi} = 1$.

You may also write W_N simply as W.

Then

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, k = 0, 1, \cdots, N-1$$

Note that $\sum_{n=0}^{N-1} (e^{\frac{-j2\pi k}{N}})^n = \sum_{n=0}^{N-1} W^{kn}$

Also Note that $\sum_{k=0}^{N-1} W_N^{k(l-n)} = \begin{cases} N, & l = n \\ 0, & l \neq n \end{cases}$

Relationship between Exponential Forms and Twiddle Factors (W) for Periodicity = N

Sr. No.	Exponential Form	Symbolic Form
01	$e^{-j2\pi n/N} = e^{-j2\pi(n+N)/N}$	$W_N^n = W_N^{n+N}$
02	$e^{-j2\pi(n+N/2)/N} = -e^{-j2\pi n/N}$	$W_N^{n+N/2} = -W_N^n$
03	$e^{-j2\pi k} = e^{-j2\pi Nk/N} = 1$	$W_N^{N+K} = 1$
04	$e^{-j2(2\pi/N)} = e^{-j2\pi/(N/2)}$	$W_N^2 = W_{N/2}$

Values of W_N^n for Period (Length) N=8

n	$W_N^n = e^{(-j2\pi)(n/N)}$	Remarks
0	$1 = 1\angle 0$	W_N^0
1	$(1-j)/\sqrt{2} = 1\angle -45°$	W_N^1
2	$-j = 1\angle -90°$	W_N^2
3	$-(1+j)/\sqrt{2} = 1\angle -135°$	W_N^3
4	$-1 = 1\angle -180°$	$W_N^4 = -W_N^0$
5	$-(1-j)/\sqrt{2} = 1\angle -225°$	$W_N^5 = -W_N^1$
6	$j = 1\angle 90°$	$W_N^6 = -W_N^2$
7	$(1+j)/\sqrt{2} = 1\angle 45°$	$W_N^7 = -W_N^3$

Matrix Relations for Direct DFT Computation

The DFT of x[n] is defined by:

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \quad 0 \le k \le N-1$$

DFT of x[n] can be expressed in NxN matrix as:

where

$$[\mathrm{X(k)}] = \left[\sum_{\mathrm{n}=0}^{N-1} W_N^{nk}\right][\mathrm{x(n)}]$$

$$\mathbf{X} = \begin{bmatrix} X[0] & X[1] & \cdots\cdots & X[N-1] \end{bmatrix}^T$$

$$\mathbf{x} = \begin{bmatrix} x[0] & x[1] & \cdots\cdots & x[N-1] \end{bmatrix}^T$$

Matrix Relations for Direct DFT Computation

$\sum_{k=0}^{N-1} W_N^{nk}$ can be expanded as NXN **DFT Matrix**

$$\sum_{k=0}^{N-1} W_N^{nk} = \begin{bmatrix} 1 & 1 & 1 & \cdots\cdots & 1 \\ 1 & W_N^1 & W_N^2 & \cdots\cdots & W_N^{(N-1)} \\ 1 & W_N^2 & W_N^4 & \cdots\cdots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & W_N^{(N-1)} & W_N^{2(N-1)} & \cdots\cdots & W_N^{(N-1)^2} \end{bmatrix}$$

DFT of x(n)=

$$X(k) := \sum_{n=0}^{n-1} \left[x(n) \left(W_N \right)^{nk} \right]$$

For N = 4, the range of n, k = [0 1 2 3] each.

Hence $X(n) = x(0)W_N^{n.0} + x(1)W_N^{n.1} + x(2)W_N^{n.2} + x(3)W_N^{n.3}$

		x(0)	x(1)	x(2)	x(3)
X(0)	=	W_4^{0x0}	W_4^{0x1}	W_4^{0x2}	W_4^{0x3}
X(1)	=	W_4^{1x0}	W_4^{1x1}	W_4^{1x2}	W_4^{1x3}
X(2)	=	W_4^{2x0}	W_4^{2x1}	W_4^{2x2}	W_4^{2x3}
X(3)	=	W_4^{3x0}	W_4^{3x1}	W_4^{3x2}	W_4^{3x3}

		x(0)	x(1)	x(2)	x(3)
X(0)	=	W_4^{0x0}	W_4^{0x1}	W_4^{0x2}	W_4^{0x3}
X(1)	=	W_4^{1x0}	W_4^{1x1}	W_4^{1x2}	W_4^{1x3}
X(2)	=	W_4^{2x0}	W_4^{2x1}	W_4^{2x2}	W_4^{2x3}
X(3)	=	W_4^{3x0}	W_4^{3x1}	W_4^{3x2}	W_4^{3x3}

		x(0)	x(1)	x(2)	x(3)
X(0)	=	W_4^{0}	W_4^{0}	W_4^{0}	W_4^{0}
X(1)	=	W_4^{0}	W_4^{1}	W_4^{2}	W_4^{3}
X(2)	=	W_4^{0}	W_4^{2}	W_4^{4}	W_4^{6}
X(3)	=	W_4^{0}	W_4^{3}	W_4^{6}	W_4^{9}

		x(0)	x(1)	x(2)	x(3)
X(0)	=	W_4^0	W_4^0	W_4^0	W_4^0
X(1)	=	W_4^0	W_4^1	W_4^2	W_4^3
X(2)	=	W_4^0	W_4^2	W_4^4	W_4^6
X(3)	=	W_4^0	W_4^3	W_4^6	W_4^9

		x(0)	x(1)	x(2)	x(3)
X(0)	=	W_4^0	W_4^0	W_4^0	W_4^0
X(1)	=	W_4^0	W_4^1	W_4^2	$-W_4^1$
X(2)	=	W_4^0	W_4^2	W_4^0	W_4^2
X(3)	=	W_4^0	$-W_4^1$	W_4^2	W_4^1

		x(0)	x(1)	x(2)	x(3)
X(0)	=	W_4^0	W_4^0	W_4^0	W_4^0
X(1)	=	W_4^0	W_4^1	W_4^2	$-W_4^1$
X(2)	=	W_4^0	W_4^2	W_4^0	W_4^2
X(3)	=	W_4^0	$-W_4^1$	W_4^2	W_4^1

		x(0)	x(1)	x(2)	x(3)
X(0)	=	1	1	1	1
X(1)	=	1	-j	-1	j
X(2)	=	1	-1	1	-1
X(3)	=	1	j	-1	-j

		x(0)	x(1)	x(2)	x(3)
X(0)	=	1	1	1	1
X(1)	=	1	-j	-1	j
X(2)	=	1	-1	1	-1
X(3)	=	1	j	-1	-j

$$\begin{pmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{pmatrix} := \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix} \cdot \begin{pmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{pmatrix}$$

Characteristic (System) Matrix for DFT

Inverse Discrete Fourier Transforms (IDFT)

- Inverse DFT of $X(k)$ (i.e. $X(0)$ to $X(N\text{-}1)$) can perfectly reconstruct time-domain signal $x(n)$ (i.e. $x(0)$ to $x(N\text{-}1)$).

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

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

Matrix Relations for IDFT Computation

Likewise, the IDFT is given by

$$x[n]=\frac{1}{\mathrm{N}}\sum_{k=0}^{N-1} X[k]W_N^{-kn},\ 0\le n\le N-1$$

IDFT of x[n] can be expressed in NxN matrix form as:

$$\mathrm{x[n]}=\left[\sum_{n=0}^{N-1} W_N^{nk}\right]^{-1}[\mathrm{X(k)}]$$

Matrix Relations for IDFT Computation

$\sum_{k=0}^{N-1} W_N^{-nk}$ can also be expanded as NXN **DFT matrix.**

$$\sum_{k=0}^{N-1} W_N^{-nk} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N^{-1} & W_N^{-2} & \cdots & W_N^{-(N-1)} \\ 1 & W_N^{-2} & W_N^{-4} & \cdots & W_N^{-2(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & W_N^{-(N-1)} & W_N^{-2(N-1)} & \cdots & W_N^{-(N-1)^2} \end{bmatrix}$$

$$\sum_{k=0}^{N-1} W_N^{-nk} = \left[1/N \sum_{n=0}^{N-1} W_N^{nk} \right]^*$$

The inversion can be done by Hermitian Conjugating j by –j and dividing by *N*.

		x(0)	x(1)	x(2)	x(3)
X(0)	=	1	1	1	1
X(1)	=	1	-j	-1	j
X(2)	=	1	-1	1	-1
X(3)	=	1	j	-1	-j

$$\begin{pmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{pmatrix} := \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix} \cdot \begin{pmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{pmatrix}$$

Characteristic (System) Matrix for DFT

$$\begin{pmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{pmatrix} := \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix} \cdot \begin{pmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{pmatrix}$$

The effective determinant of the above Characteristic Matrix is 1/4.

Conversion in Characteristic Matrices of DFT/IDFT done by replacing j by –j.

Inversion of the above characteristic matrix is given as follows:

$$\begin{pmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{pmatrix} := \left[\frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{pmatrix} \cdot \begin{pmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{pmatrix} \right]$$

CHARECTERISTIC MATRIX OF 8-POINT DFT (N=8)

W_8^{0x0}	W_8^{0x1}	W_8^{0x2}	W_8^{0x3}	W_8^{0x4}	W_8^{0x5}	W_8^{0x6}	W_8^{0x7}
W_8^{1x0}	W_8^{1x1}	W_8^{1x2}	W_8^{1x3}	W_8^{1x4}	W_8^{1x5}	W_8^{1x6}	W_8^{1x7}
W_8^{2x0}	W_8^{2x1}	W_8^{2x2}	W_8^{2x3}	W_8^{2x4}	W_8^{2x5}	W_8^{2x6}	W_8^{2x7}
W_8^{3x0}	W_8^{3x1}	W_8^{3x2}	W_8^{3x3}	W_8^{3x4}	W_8^{3x5}	W_8^{3x6}	W_8^{3x7}
W_8^{4x0}	W_8^{4x1}	W_8^{4x2}	W_8^{4x3}	W_8^{4x4}	W_8^{4x5}	W_8^{4x6}	W_8^{4x7}
W_8^{5x0}	W_8^{5x1}	W_8^{5x2}	W_8^{5x3}	W_8^{5x4}	W_8^{5x5}	W_8^{5x6}	W_8^{5x7}
W_8^{6x0}	W_8^{6x1}	W_8^{6x2}	W_8^{6x3}	W_8^{6x4}	W_8^{6x5}	W_8^{6x6}	W_8^{6x7}
W_8^{7x0}	W_8^{7x1}	W_8^{7x2}	W_8^{7x0}	W_8^{7x4}	W_8^{7x5}	W_8^{7x6}	W_8^{7x7}

Characteristic Matrix of 8-Point DFT (N=8)

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & \frac{1-j}{\sqrt{2}} & -j & \frac{-(1+j)}{\sqrt{2}} & -1 & \frac{-(1-j)}{\sqrt{2}} & j & \frac{1+j}{\sqrt{2}} \\
1 & -j & -1 & j & 1 & -j & -1 & j \\
1 & \frac{-(1+j)}{\sqrt{2}} & j & \frac{1-j}{\sqrt{2}} & -1 & \frac{1+j}{\sqrt{2}} & -j & \frac{-(1-j)}{\sqrt{2}} \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
1 & \frac{-(1-j)}{\sqrt{2}} & -j & \frac{1+j}{\sqrt{2}} & -1 & \frac{1-j}{\sqrt{2}} & j & \frac{-(1+j)}{\sqrt{2}} \\
1 & j & -1 & -j & 1 & j & -1 & -j \\
1 & \frac{1+j}{\sqrt{2}} & j & \frac{-(1-j)}{\sqrt{2}} & -1 & \frac{-(1+j)}{\sqrt{2}} & -j & \frac{(1-j)}{\sqrt{2}}
\end{bmatrix}$$

DFT and Inverse DFT

Conversion in Characteristic (System) Matrices of DFT and IDFT can be obtained by replacing j by –j.

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & \frac{1-j}{\sqrt{2}} & -j & \frac{-(1+j)}{\sqrt{2}} & -1 & \frac{-(1-j)}{\sqrt{2}} & j & \frac{1+j}{\sqrt{2}} \\
1 & -j & -1 & j & 1 & -j & -1 & j \\
1 & \frac{-(1+j)}{\sqrt{2}} & j & \frac{1-j}{\sqrt{2}} & -1 & \frac{1+j}{\sqrt{2}} & -j & \frac{-(1-j)}{\sqrt{2}} \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
1 & \frac{-(1-j)}{\sqrt{2}} & -j & \frac{1+j}{\sqrt{2}} & -1 & \frac{1-j}{\sqrt{2}} & j & \frac{-(1+j)}{\sqrt{2}} \\
1 & j & -1 & -j & 1 & j & -1 & -j \\
1 & \frac{1+j}{\sqrt{2}} & j & \frac{-(1-j)}{\sqrt{2}} & -1 & \frac{-(1+j)}{\sqrt{2}} & -j & \frac{(1-j)}{\sqrt{2}}
\end{bmatrix}$$

System Matrix (DFT)

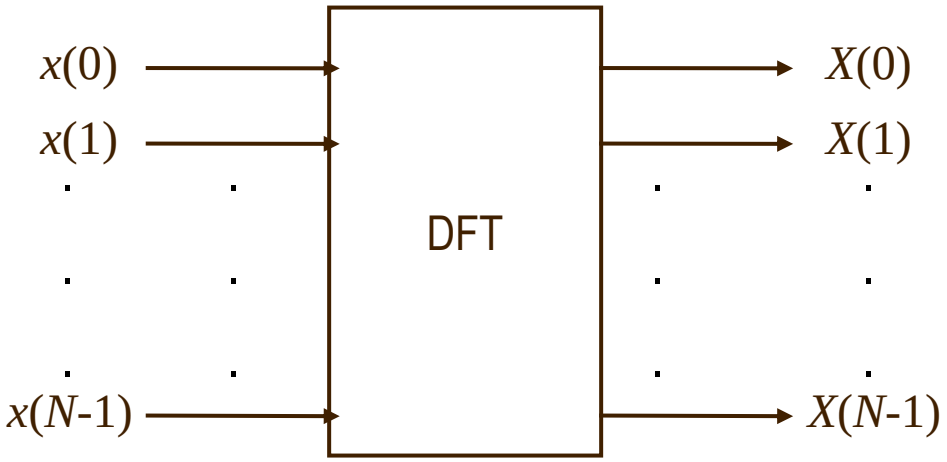
Effective Determinant

$$\frac{1}{8}\cdot\left(\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & \frac{1+j}{\sqrt{2}} & j & -\frac{1-j}{\sqrt{2}} & -1 & -\frac{1+j}{\sqrt{2}} & -j & \frac{1-j}{\sqrt{2}} \\
1 & j & -1 & -j & 1 & j & -1 & -j \\
1 & -\frac{1-j}{\sqrt{2}} & -j & \frac{1+j}{\sqrt{2}} & -1 & \frac{1-j}{\sqrt{2}} & j & -\frac{1+j}{\sqrt{2}} \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
1 & -\frac{1+j}{\sqrt{2}} & j & \frac{1-j}{\sqrt{2}} & -1 & \frac{1+j}{\sqrt{2}} & -j & -\frac{1-j}{\sqrt{2}} \\
1 & -j & -1 & j & 1 & -j & -1 & j \\
1 & \frac{1-j}{\sqrt{2}} & -j & -\frac{1+j}{\sqrt{2}} & -1 & -\frac{1-j}{\sqrt{2}} & j & \frac{1+j}{\sqrt{2}}
\end{bmatrix}\right)$$

System Matrix (IDFT)

DFT Properties

- N-point DFT maps
  - N inputs in the time domain: $x(0)$ through $x(N\text{-}1)$ to
  - N output in the frequency domain: $X(0)$ to $X(N\text{-}1)$
  - Inherent block processing
- If $x(n)$ is real then $X(N\text{-}k) = X^*(k)$ for $k=1$ to $N/2\text{-}1$ where * denotes the complex conjugate
  - $|X(k)|$ is symmtric about $k=N/2$
  - Even if $x(n)$ is real, $X(k)$ is usually complex (except for $X(0)$ and $X(N/2)$)
- $X(0)$ denotes DC frequency, $X(N/2)$ denotes $Fs/2$ frequency



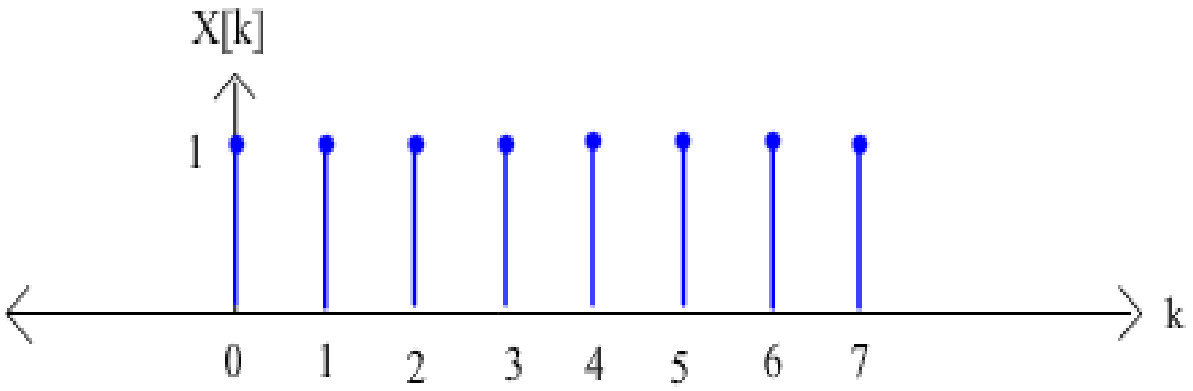
Example of DFT

$$x[n] = \begin{cases} 1, & n = 0 \\ 0, & n = 1, \ldots, 7 \end{cases}$$

- Find X[k]

  We know that k=1,.., 7; N=8

$$X[k] = \sum_{n=0}^{7} x[n] W_N^{kn} = \sum_{n=0}^{7} \delta[n] W_N^{kn} = 1 \quad , \forall\, k$$



Example of DFT

Given $y[n] = \delta[n-2]$ and $N = 8$, find $Y[k]$.

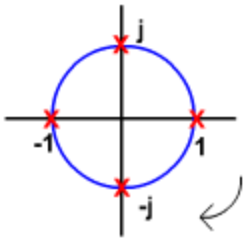
$$Y[k] = \sum_{n=0}^{7} \delta[n-2] W_N^{kn} = W_N^{2k} = e^{\frac{-j2\pi 2k}{N}} = e^{\frac{-j\pi k}{2}}$$

because N = 8

$$= (-j)^k$$

$$Y[k] = [1, -j, -1, j, 1, -j, -1, j]$$

$$x[n-n_0] \longleftrightarrow W_N^{nok} X[k]$$



Polar plot for W_N^{2k}

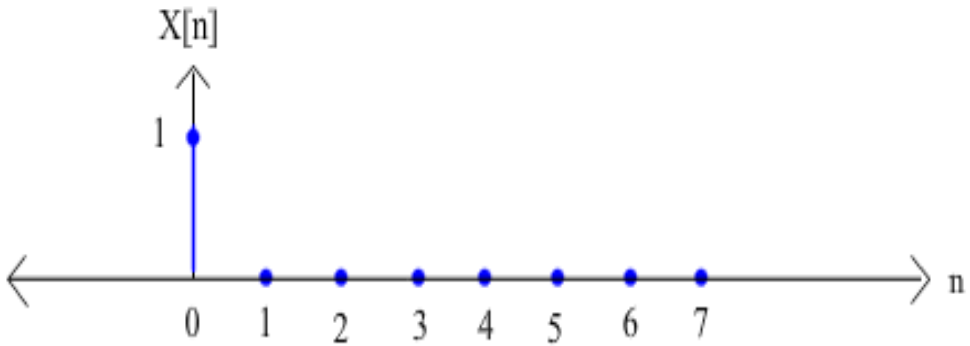
Time Shift Property of DFT

$$x[n-n_0]_{\text{mod } N} \leftrightarrow W_N^{kn_0} X[k]$$

Example of IDFT

Find the IDFT of $X[k] = 1, k = 0, 1, \ldots, 7.$

$$x[n] = \frac{1}{8}\sum_{n=0}^{7} W_N^{-kn} = \frac{1}{8} N\,\delta[n] = \delta[n]$$



Remember:

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]W_N^{-kn},\ n = 0, 1, \ldots, N-1$$

QUIZ: DFT and IDFT Problems

- Compute the DFT of the following. Also Verify your answers (DFTs) by applying IDFT:

A) $x(n) = [1 \quad \underset{\uparrow}{3} \quad 2 \quad 1]$ $h(n) = [\underset{\uparrow}{1} \quad 1]$

B) $x(n) = [\underset{\uparrow}{1} \quad 2 \quad 3 \quad 4 \quad 5]$ $h(n) = [\underset{\uparrow}{1} \quad -1]$

C) $x(n) = [2 \quad \underset{\uparrow}{1} \quad 3 \quad 2 \quad 1]$ $h(n) = [\underset{\uparrow}{4} \quad 3 \quad 2 \quad 1]$

D) $x(n) = [1 \quad \underset{\uparrow}{1} \quad 1]$ $h(n) = [\underset{\uparrow}{1} \quad 0.5 \quad 0.25]$

E) $x(n) = [\underset{\uparrow}{1} \; 0 \; 1 \; 0 \; 1 \; 1 \; 1 \; 1]$ $h(n) = [\underset{\uparrow}{1} \quad 1 \quad 1]$

F) $x(n) = [\underset{\uparrow}{3} \quad 2 \quad 1 \quad 0]$ $h(n) = [1 \quad \underset{\uparrow}{1} \quad 1]$

G) $x(n) = [\underset{\uparrow}{3} \quad 2 \quad 1 \quad 0 \quad 0 \quad 0]$ $h(n) = [1 \quad \underset{\uparrow}{1} \quad 1]$

Using MATLAB to Calculate DFT

- Example:
  - Assume N=4
  - x[n]=[1,2,3,4]
  - n=0,…,3
  - Find X[k]; k=0,…,3

```
1 -  N=4;
2 -  x=[1,2,3,4]
3 -  for k1=1:N
4 -      X(k1)=0;
5 -      k=k1-1;
6 -      for n1=1:N;
7 -          n=n1-1;
8 -          X(k1)=X(k1)+x(n1)*exp(-j*2*pi*k*n/N);
9 -      end
10 - end
11 - x
12 - X
```

```
X =

  10.0000            -2.0000 + 2.0000i  -2.0000 - 0.0000i  -2.0000 - 2.0000i
```

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, k = 0, 1, \cdots, N-1$$

or

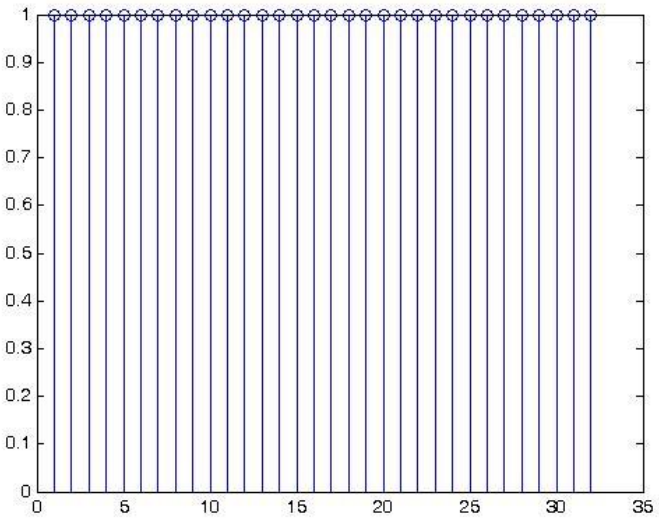
$$\sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi kn}{N}}, k = 0, 1, \cdots, N-1$$

Matlab Example: f=0 (DC), N=32-point DFT

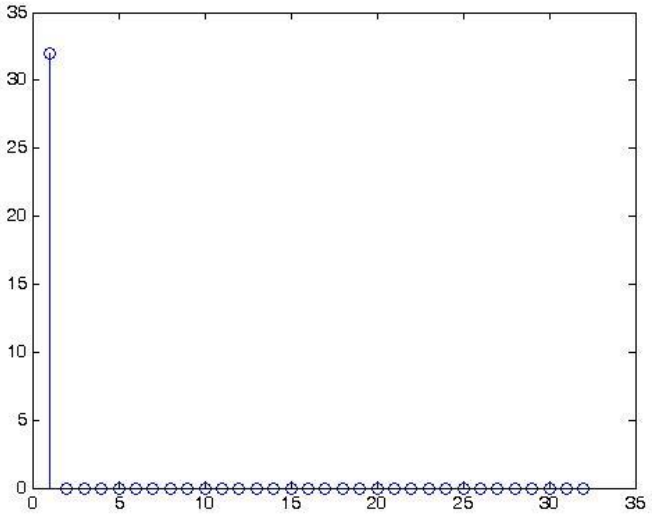
```
n=0:31;
f=0.0;
x = cos(2*pi*n*f);
figure(1); stem(x);
figure(2); stem(abs(fft(x)));
```

Single peak at X(0) = 0

x(n)



|X(k)|



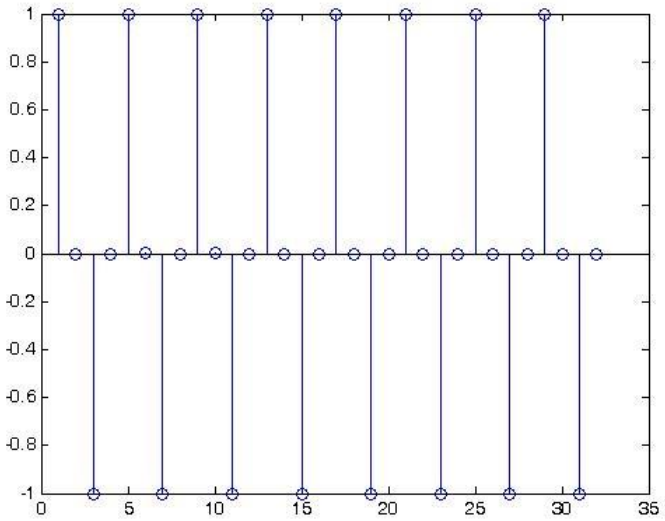
NOTE: Matlab indexes starting from 1 (i.e x(1) to x(N)) not 0 (i.e. x(0) to x(N-1))

Matlab Example: f=0.25, N=32-point DFT

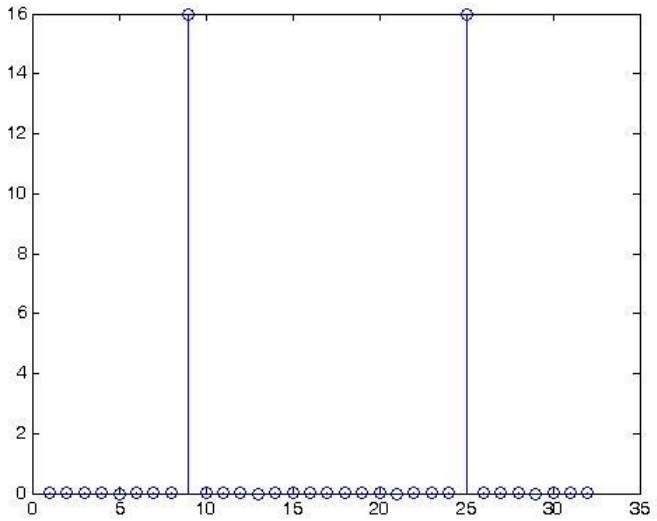
```
n=0:31;
f=0.25;
x = cos(2*pi*n*f);
figure(1); stem(x);
figure(2); stem(abs(fft(x)));
```

Peaks at X(N/4) and X(3N/4)

x(n)



|X(k)|



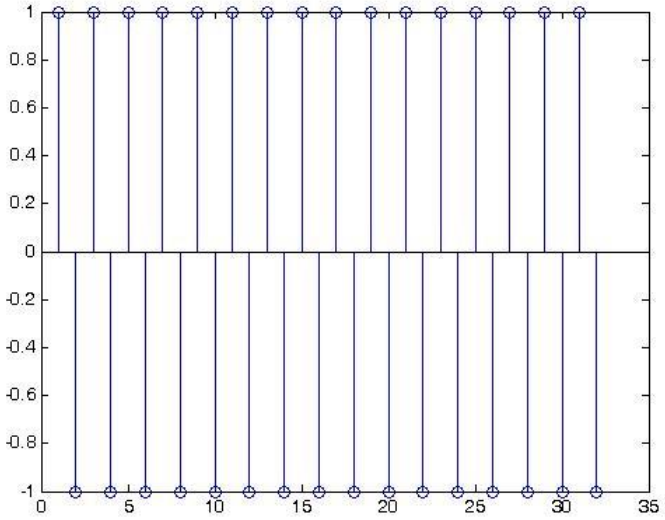
NOTE: Matlab indexes starting from 1 (i.e x(1) to x(N)) not 0 (i.e. x(0) to x(N-1))

Matlab Example: f=0.5(analog f = fs/2), N=32-point DFT

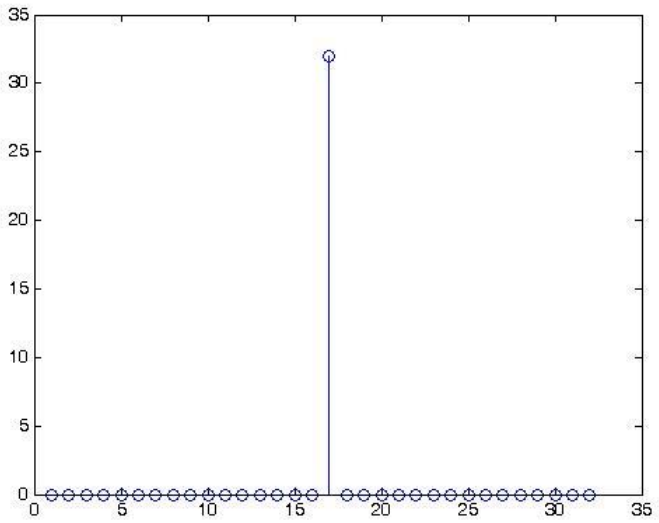
```
n=0:31;
f=0.5;
x = cos(2*pi*n*f);
figure(1); stem(x);
figure(2); stem(abs(fft(x)));
```

Single peak at X(N/2)

x(n)



|X(k)|



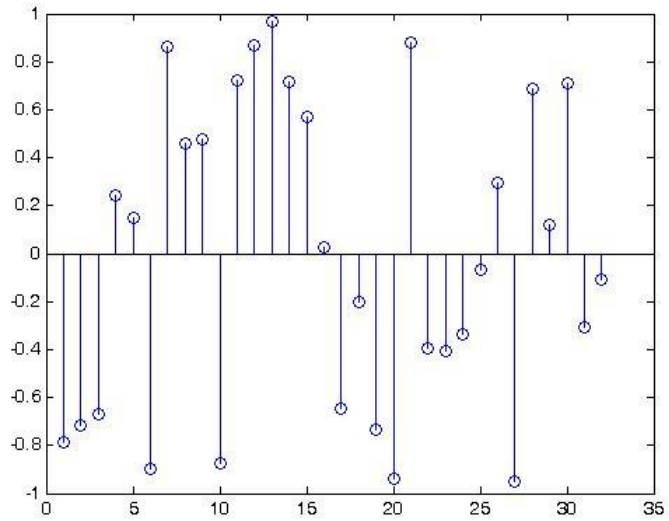
NOTE: Matlab indexes starting from 1 (i.e x(1) to x(N)) not 0 (i.e. x(0) to x(N-1))

Matlab Example: Random Input, , N=32-point DFT

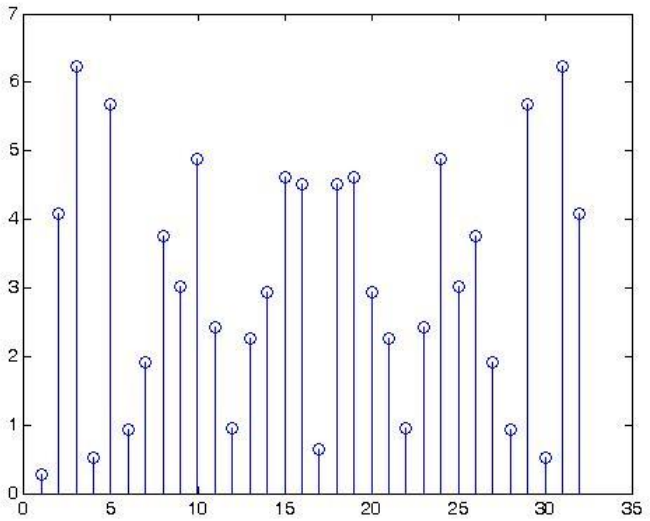
```
n=0:31;
x = -1 + (1 - -1).*rand(length(n),1);
figure(1); stem(x);
figure(2); stem(abs(fft(x)));
```

Random frequency content
(since x(n) real, X(k) symmetric about X(N/2))

x(n)



|X(k)|



NOTE: Matlab indexes starting from 1 (i.e x(1) to x(N)) not 0 (i.e. x(0) to x(N-1))

Discrete Fourier Transform Computations

- The DFT equation:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \qquad k = 0,1,\ldots,N-1 \qquad W_N = e^{-j2\pi/N}$$

- W_N's are also called “Twiddle Factors" which are complex values around the Unit Circle in the Complex Plane
  - Multiplication by twiddle factor serve to "rotate" a value around the unit circle in the complex plane
- Computations:
  - To compute $X(0)$, require N Complex Multiplications
  - To compute $X(1)$, require N Complex Multiplications
  - ..To compute $X(N\text{-}1)$, require N Complex Multiplications
- Total: N^2 Complex Multiplications and $N(N\text{-}1) = N^2 - N$ Complex Additions to compute N-point DFT

Example DFT Applications

- Correlation Analysis
- Data Compression (Audio - MP3, Image – JPEG, Video – MP4)
- Frequency-Domain Filtering (Convolution Operations)
- Implementation of MCM (multi-carrier modulation) and OFDM (Orthogonal Frequency Division Multiplexing) commn. systems
- Interpolation
- Medical Imaging (Healthcare Informatics)
- Seismic Tremor Analysis
- Spectral Estimation
- Speech Recognition/Synthesis
- --- Many More Applications ---

FFT Applications (OFDM)

- An Orthogonal Frequency Division Multiplexing (OFDM) signal consists of a sum of subcarriers that are modulated by using the Phase Shift Keying (PSK) technique or Quadrature Amplitude Modulation (QAM) technique.
- OFDM technique is widely used for high-speed digital Commns (xDSL, DAB, DVB-T/H, and WLAN). In OFDM, Discrete Fourier Transform (DFT)/Inverse-DFT is used, as it is an important operation. DFT/IDFT computation requires a large amount of arithmetic operations, hence we need an efficient FFT algorithm (Radix-8/4/2 FFT) which can reduce the number of arithmetic operations to meet real time computation in OFDM systems.

FFT Applications (OFDM) (contd...)

- For more information on *IEEE 802.16, refer to the IEEE Standard for Local and Metropolitan Area Networks.* OFDM is one of the key physical layer components associated with mobile worldwide interoperability for microwave access (WiMax) and is widely regarded as an enabling technology for future broadband wireless protocols including 3GPP and 3GPP2 long term evolution standards.

- The OFDM system kernel has the following salient features: Support for 128, 512, 1K, and 2K FFT sizes to address variable bandwidths from 1.25 to 20 MHz.

FFT Applications (Healthcare Informatics)

- Medical Image Reconstruction with the help of FFT
- In a number of medical imaging modalities, the 2D FFT algorithm is being used for the reconstruction of images from data in magnetic resonance imaging (MRI) and ultrasonic scanning imaging.
- In medical imaging like computed tomography (CT), MRI, and ultrasonic scanning, among others, very specialized hardware or CPUs are used to reconstruct the images from acquired raw data. GPUs have the potential for providing better performance compared to the CPUs in medical image reconstruction at a cheaper cost.

FFT Applications (Speech Recognition)

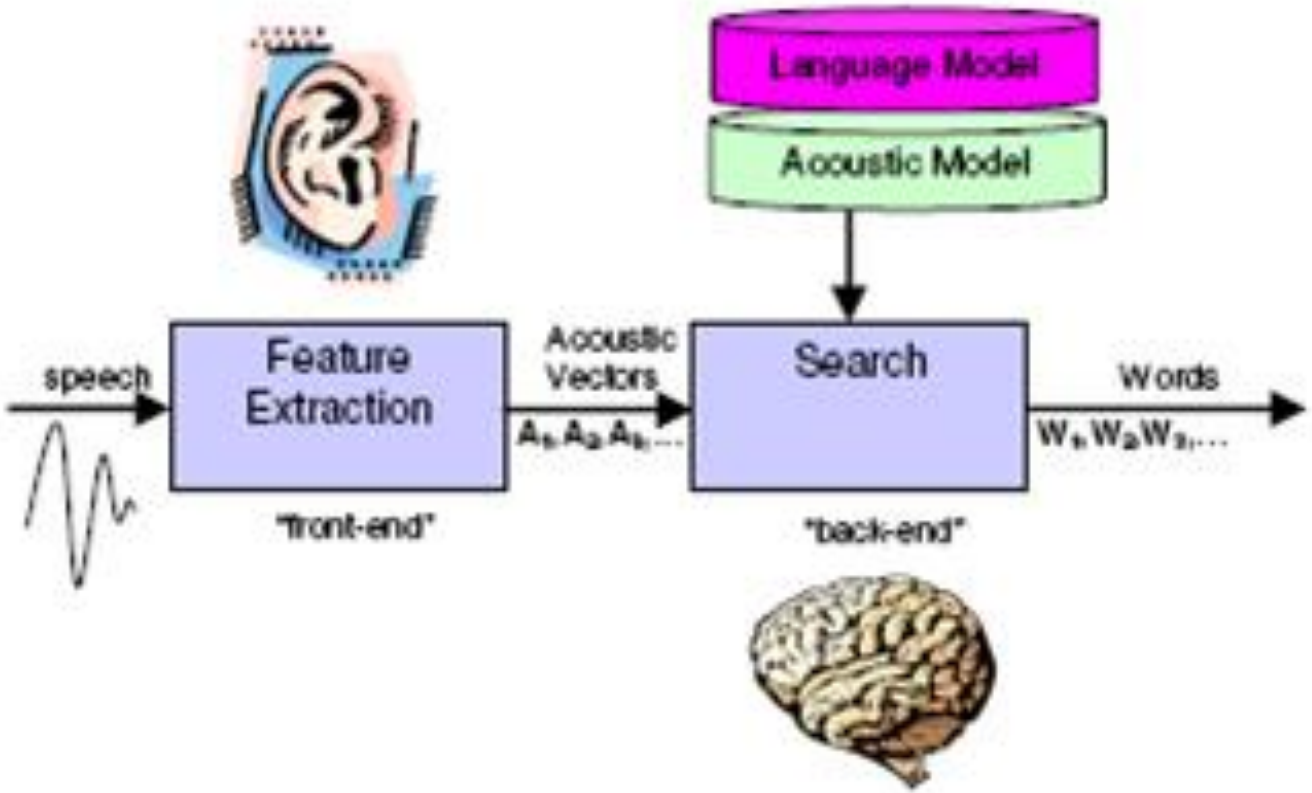


Figure: - Basic structure of an ASR system.

Is FFT Better than DFT?

DFT (Walking Speed)

- Why is this important? Where is this used?
  - allows machines to calculate the frequency domain
  - allows for the convolution of signals by just multiplying them together
  - Used in digital spectral analysis for speech, imaging and pattern recognition as well as signal compression and signal manipulation using filters

- But the DFT requires $\mathbf{N^2}$ Complex Multiplications and requires $\mathbf{N(N-1) = N^2 - N}$ Complex Additions.

Discrete Fourier Transforms (Contd…)

- Forward (Direct) Transform

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk}$$

for k = 0, 1, …, N-1

Exponent of W_N has Period N

- Memory Usage

$x[n]$: N complex words of RAM

$X[k]$: N complex words of RAM

W_N : N complex words of ROM

- Reduce Memory Usage by 50%

Allow Output Array $X[k]$ to write over Input Array $x[n]$

Exploit Twiddle Factors Symmetry

- Computational Complexity

N^2 Complex Multiplications

N (N –1) Complex Additions

N^2 Integer Multiplications

N^2 Modulo Indexes into Lookup Table of Twiddle Factors

- Inverse (Backward) Transform

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{-nk}$$

for n = 0, 1, …, N-1

Memory Usage?

Computational Complexity?

FFT (Jet Speed)

- J. W. Cooley and J. W. Tukey developed the FFT in 1965
- FFT algorithm for **more** efficiently computing the DFT
- Exploiting Symmetry and Periodicity in Twiddle Factors and uses a Divide and Conquer Strategy for reducing the Time Complexity for Computing the DFT Coefficients
- Symmetry: $W_N^{r+N/2} = -W_N^r$
- Periodicity: $W_N^{r+N} = W_N^r$
- Requires only $\mathbf{N.Log_2(N)}$ Complex Additions and $\mathbf{(N/2).Log_2(N)}$ Complex Multiplications!
  - Faster computation times
  - More precise results due to less round-off error

Computation Advantage of Radix-2 FFT Algorithms for Various Values of N

Number of Computation Points	DFT Computations			FFT Computatons			Improvement (Speedup)
	Complex Multiplications	Complex Additions	Total	Complex Multiplications	Complex Additions	Total	
N	N^2	N^2-N	A	(N/2).Log(N)	N.Log(N)	B	A/B
2	4	2	6	1.00	2.00	3.00	2.00
4	16	12	28	4.00	8.00	12.00	2.33
8	64	56	120	12.00	24.00	36.00	3.33
16	256	240	496	32.00	64.00	96.00	5.17
32	1024	992	2016	80.00	160.00	240.00	8.40
64	4096	4032	8128	192.00	384.00	576.00	14.11
128	16384	16256	32640	448.00	896.00	1344.00	24.29
256	65536	65280	130816	1024.00	2048.00	3072.00	42.58
512	262144	261632	523776	2304.00	4608.00	6912.00	75.78
1024	1048576	1047552	2096128	5120.00	10240.00	15360.00	136.47
2048	4194304	4192256	8386560	11264.00	22528.00	33792.00	248.18
4096	16777216	16773120	33550336	24576.00	49152.00	73728.00	455.06
8192	67108864	67100672	134209536	53248.00	106496.00	159744.00	840.15
16384	268435456	268419072	536854528	114688.00	229376.00	344064.00	1560.33

FFT Algorithms

- Several different types of FFT Algorithms (Radix-2, Radix-4, Radix-8, Decimation in Time (DIT) (DIT FFT Algorithm) and Decimation in Frequency (DIF) (DIF FFT Algorithm)
- Focus on Radix-2 using DIT FFT Algorithms
  - Breaks down DFT calculation into number of 2-point DFTs
  - Each 2-point DFT uses an operation called the Butterfly
  - These groups are then re-combined with another group of two and so on for number of stages $M = \log_2(N)$
  - Using the DIT method the given input sample points must be reordered using the bit reversal logic

FFT Algorithms

- Communication system application: multicarrier modulation using harmonically related carriers
  - Discrete multi-tone modulation in ADSL & VDSL modems
  - OFDM Transceivers such as in IEEE 802.11a wireless LANs
- Efficient Divide-and-Conquer Algorithm
  - Compute Discrete Fourier Transform of length $N = 2^M$
  - ½ $N \log_2 N$ Complex Multiplications and $N \log_2 N$ Additions
  - How many real complex multiplications and additions?
- Derivation: Assume N is even and power of two

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk} = \sum_{n=even}^{N-1} x[n] W_N^{nk} + \sum_{n=odd}^{N-1} x[n] W_N^{nk}$$

FFT Algorithms: Divide-and-Conquer Strategy

The most-well known algorithm design strategy:

1. Divide the instance of large problem into two or more sub-problems (smaller instances)

2. Solve the sub-problems (smaller instances) recursively

3. Obtain the solution to original (large problem) instance by combining these solutions for the smaller instances

Divide and Conquer Strategy

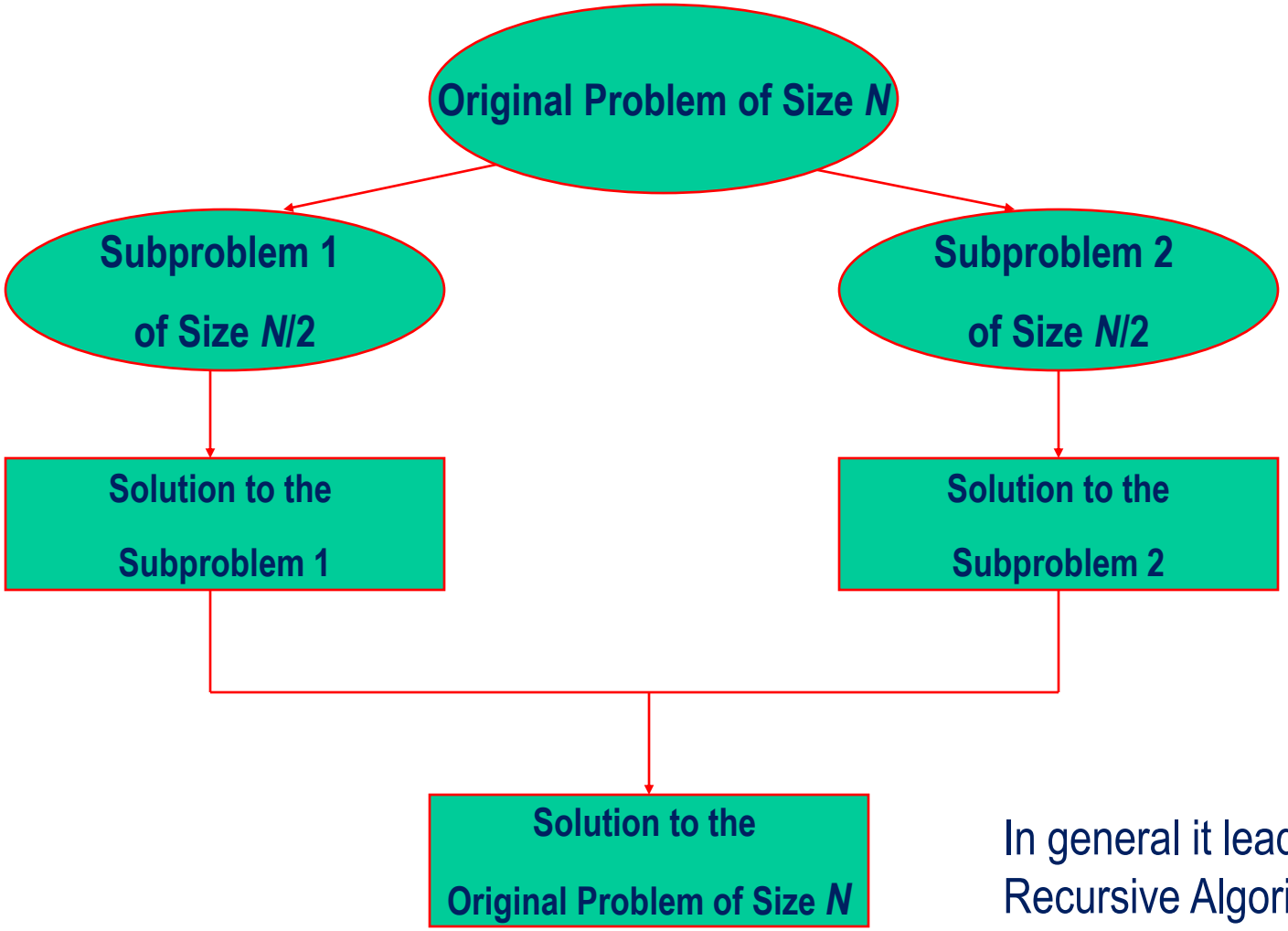
Divide-and-Conquer Strategy for an Algorithm Design:

Divide: If the input size is too large to deal with in a straightforward manner, then divide this big problem into two or more disjoint subproblems

Conquer: Use the divide and conquer strategy recursively to solve these subproblems

Combine: Merge the solutions to these subproblems into a complete solution for the original problem of big size

Divide-and-Conquer Strategy (contd...)



In general it leads to a Recursive Algorithm!

Radix 2 Decimation-in-Time (DIT) FFT Algo

- First break x[n] into even and odd parts
- Let n=2m for even part and n=2m+1 for odd part
- Even and Odd parts are DFTs of a N/2 point sequence
- Break N/2 size subsequence into two halves of N/4 size sub-subsequences by letting 2m→m
- The 1st sub-subsequence here is the term x[0], x[4], …
- The 2nd sub-subsequence here is the term x[2], x[6], …

Radix 2 DIT FFT Algorithm (contd…)

- Substitute $n = 2m$ for n even and $n = 2m+1$ for odd

$$X[k] = \sum_{m=0}^{N/2-1} x[2m]\, W_N^{2mk} + \sum_{m=0}^{N/2-1} x[2m+1]\, W_N^{(2m+1)k}$$

$$= \sum_{m=0}^{N/2-1} x[2m] \left(W_N^2\right)^{mk} + W_N^k \sum_{m=0}^{N/2-1} x[2m+1] \left(W_N^2\right)^{mk}$$

- Using the Property $W_N^{2l} = e^{-j2\pi\frac{2l}{N}} = e^{-j\frac{2\pi l}{N/2}} = W_{N/2}^l$

$$X[k] = \sum_{m=0}^{N/2-1} x[2m]\, W_{N/2}^{mk} + W_N^k \sum_{m=0}^{N/2-1} x[2m+1]\, W_{N/2}^{mk} = X_1[k] + W_N^k\, X_2[k]$$

One FFT length N => two FFTs length $N/2$

Repeat process until 2-point FFTs remain.

Computational Complexity of 2-point FFT?

(Each 2-point FFT Butterfly needs 1 Multiplier and 2 Adders)

Two-Point FFT

$$X[0] = x[0] + x[1]$$

$$X[1] = x[0] - x[1]$$

Radix-2 DIT FFT Algorithm – An Example

Let's take an example where only two points are given n=0, n=1; N=2

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

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

$$W_{N/2}^{m+N/2}=W_{N/2}^{m}W_{N/2}^{N/2}=W_{N/2}^{m}$$

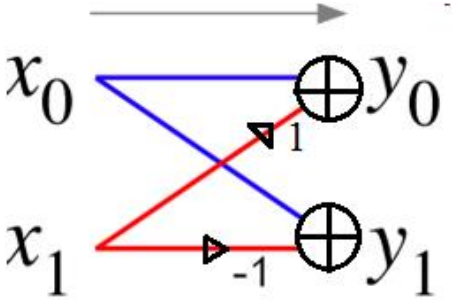
$$W_N^{N}=e^{-2\pi j}=\cos(-2\pi)-j\sin(-2\pi)=1$$

$$W_N^{N/2}=-1$$

$$X[k=0]=\sum_{m=0}^{0} W_1^{0.0}\,x[0]+W_1^{0}\left(\sum_{m=0}^{0} W_1^{0.0}\,x[1]\right)=x[0]+x[1]$$

$$X[k=1]=\sum_{m=0}^{0} W_1^{0.1}\,x[0]+W_1^{1}\left(\sum_{m=0}^{0} W_1^{0.1}\,x[1]\right)=x[0]+W_1^{1}x[1]=x[0]-x[1]$$

2-point FFT Butterfly



Where x_0=$x[0]$ and y_0= $X[k=0]$ = $X[0]$;
x_1=$x[1]$ and y_1 = $X[k=1]$ = $X[1]$

2-point FFT Butterfly

Decimation and Partition

$x_0\ x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ x_7$

Decimation (LSB Sort)

$x_0\ x_2\ x_4\ x_6$ EVEN

$x_1\ x_3\ x_5\ x_7$ ODD

Partition (MSB Sort)

$x_0\ x_1\ x_2\ x_3$ LEFT

$x_4\ x_5\ x_6\ x_7$ RIGHT

Decimation in Time $\Leftrightarrow$ Partition in Frequency

Partition in Time $\Leftrightarrow$ Decimation in Frequency

Decimation-in-Time is Partition-in-Frequency

We get further savings by exploiting the relationship between Decimation-in-Time and Partition-in-Frequency

Comparing frequency values in 2 partitions

$$X_k = \sum_{n=0}^{N-1} x_n W_N^{nk}$$

$$X_{k+\frac{N}{2}} = \sum_{n=0}^{N-1} x_n W_N^{nk} W_N^{\frac{Nn}{2}}$$

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

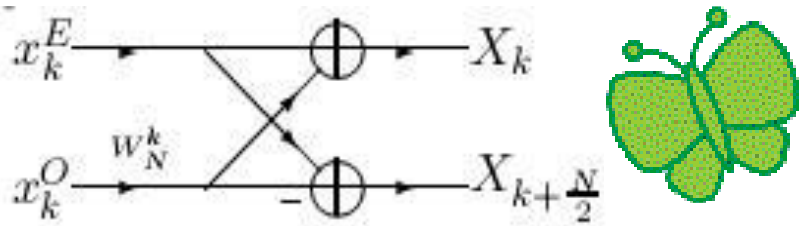
Note that same products just different signs

$+ - + - + - + -$

Using the results of the Decimation, we see that the all odd terms have **-ve** sign.

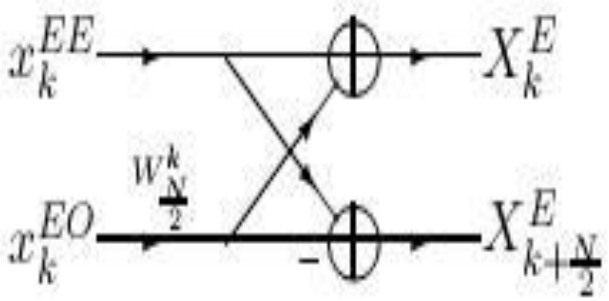
$$X_{k+\frac{N}{2}} = \sum_{n=0}^{\frac{N}{2}-1} x_n^E W_{\frac{N}{2}}^{nk} - W_N^k \sum_{n=0}^{\frac{N}{2}-1} x_n^O W_{\frac{N}{2}}^{nk}$$

Combining the two we get a basic "Butterfly"



DIT all the way

- But we needn't stop after splitting the original input sequence (of size N) into two equal parts (of size N/2)!
- Each N/2 sub-sequence can be further decimated into two equal parts (of size N/4) and this process can be continued till we get 2-point sub-sub-sequences.

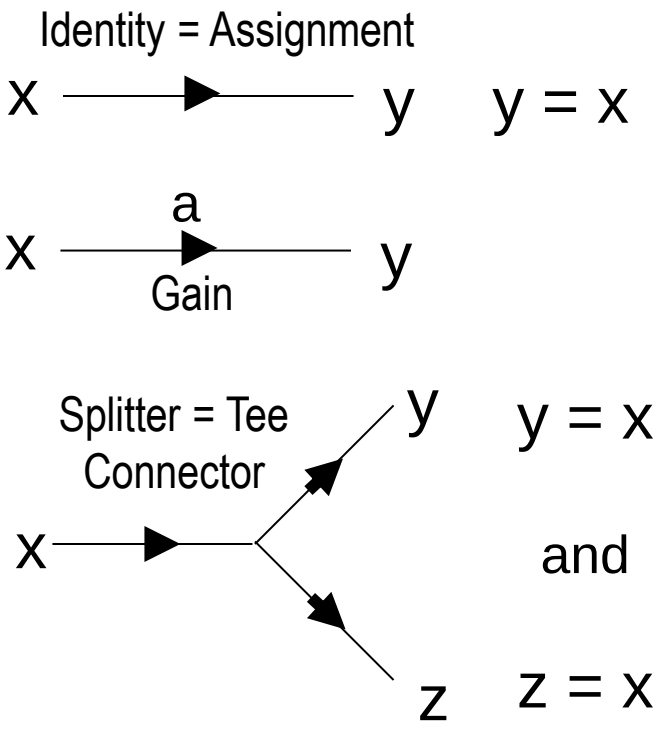


- Assuming that N is a power of 2, we will continue the process of decimation until we get basic N=2 Butterfly.

DIT all the way (contd…)

- Let N be Even No. ($N = 2^M$), then we need "$M = Log_2N$" no. of Decimation Stages for Computing the N-point FFT.
- Each Decimation Stage needs N/2 2-point Butterflies.
- Each 2-point Butterfly needs 1 Multiplier and 2 Adders.
- Total Number of **Complex Multiplications** Required for N-point DIT FFT are $1.M.(N/2) = (N/2).Log_2N$
- Further, Total Number of **Complex Additions** Required for N-point DIT FFT are $2.M.(N/2) = N.M = N.Log_2N$

Role of Graph Theory in Signal Processing

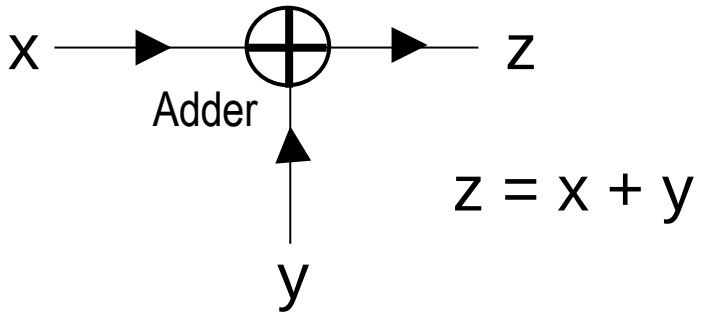


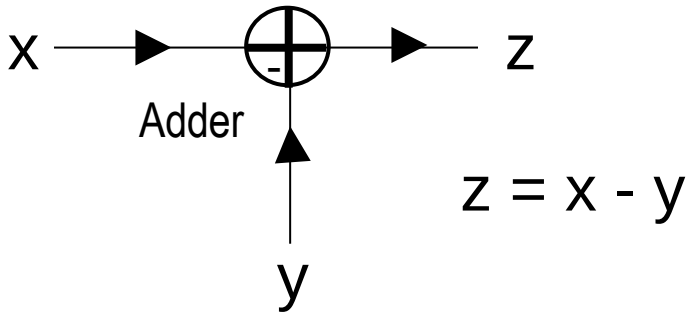
Signal-flow Graphs are made up of

- Points
- Directed Lines
- Special Symbols

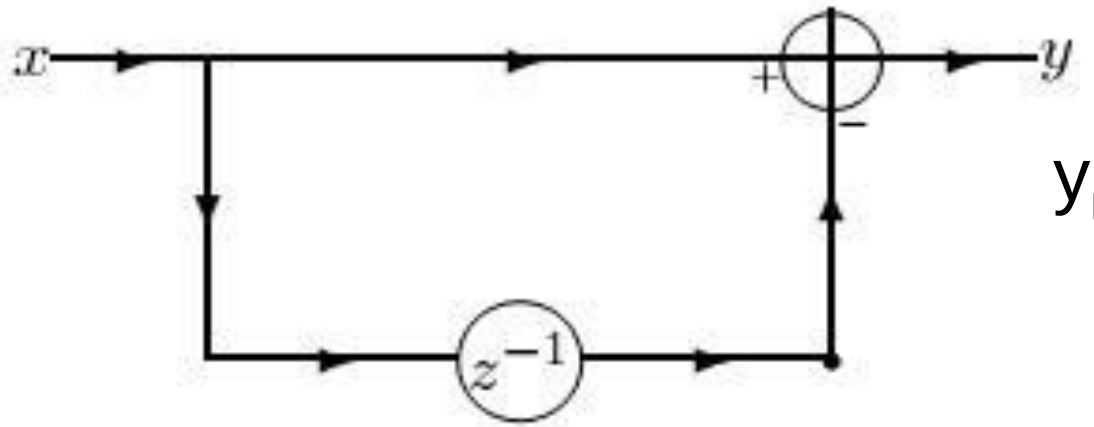
Points = Signals

All the Rest = Signal Processing Systems

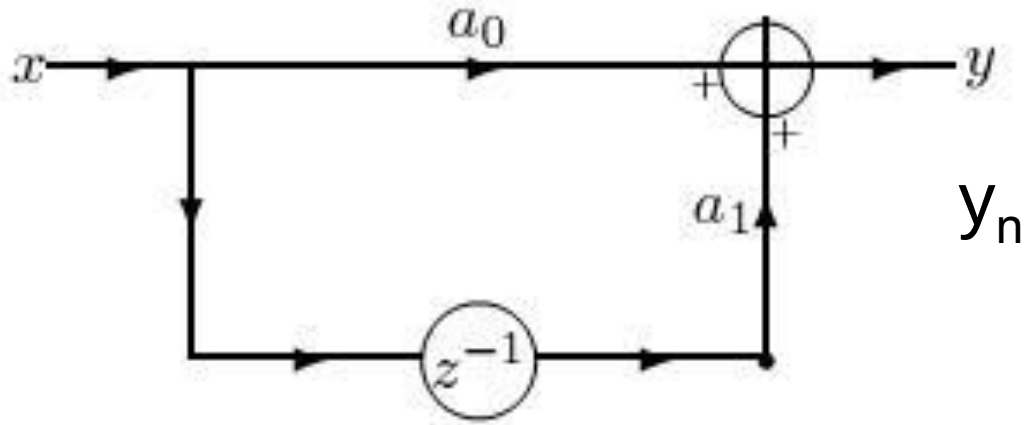




Basic Blocks for Difference Equations



$y_n = x_n - x_{n-1}$



$y_n = a_0 x_n + a_1 x_{n-1}$

Note: z^{-1} is a 1Bit Delay Element. Explicitly draw point only when need to store value (memory point)

Why is Graph Theory Useful ?

- Signal Flow Graphs capture both (i) Algorithms and (ii) Data Structures
- Graphical mechanisms for simplifying the analysis and synthesis

  (lowering Microprocessor without Interlocked Pipeline Stages (MIPS) –

  Reduced Instruction Set Computer (RISC) Architecture)
- Four basic transformations
  - Topological (move points around)
  - Commutation of Filters (any two filters commute!)
  - Unification of identical data points and removal of redundant branches
  - Transposition theorem

Fast Fourier Transforms

Discrete Fourier Transform Computations

- The DFT equation is as follows:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \qquad k = 0,1,\ldots,N-1 \qquad W_N = e^{-j2\pi/N}$$

- W_N's are also called “Twiddle Factors" which are complex values around the Unit Circle in the Complex Plane
  - Multiplication by the twiddle factor will serve to "rotate" a value around the unit circle in the complex plane
- Computations:
  - To compute $X(0)$, require N Complex Multiplications
  - To compute $X(1)$, require N Complex Multiplications
  - ..To compute $X(N\text{-}1)$, require N Complex Multiplications
- N^2 Complex Multiplications; $N(N\text{-}1) = N^2 - N$ Complex Additions to compute the N-point DFT using the straightforward method.

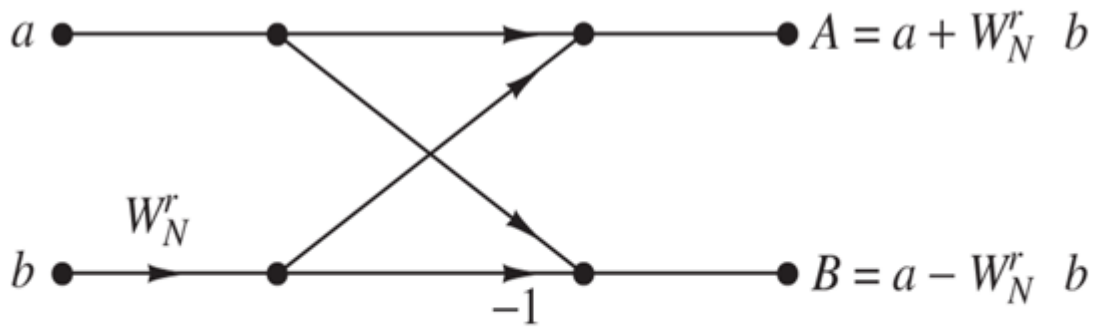
Fast Fourier Transforms

- Can exploit shared twiddle factor properties (i.e. sub-expression sharing) to reduce the number of multiplications in DFT
- These class of algorithms are called Fast Fourier Transforms
  - An FFT is simply an efficient implementation of the DFT
  - Mathematically FFT = DFT
- FFT exploits two key properties associated with the twiddle factors:
  - Symmetry Property: $W_N^{k+N/2} = -W_N^k$
  - Periodicity Property: $W_N^{k+N} = W_N^k$
- FFTs use a Divide and Conquer Algorithmic Approach, and thus we can Break or Decimate or Divide an N-point DFT into several smaller DFTs
  - N can be factored as $N=r_1r_2r_2\ldots r_v$ where the $\{r_i\}$ are prime numbers
  - Particular focus on $r_1=r_2=..=r_v=r$, where r is called the radix of FFT algorithm
  - In this case $N=r^v$ and the FFT has a regular pattern
  - We will mainly focus on Radix-2 (r=2) FFTs in this course.

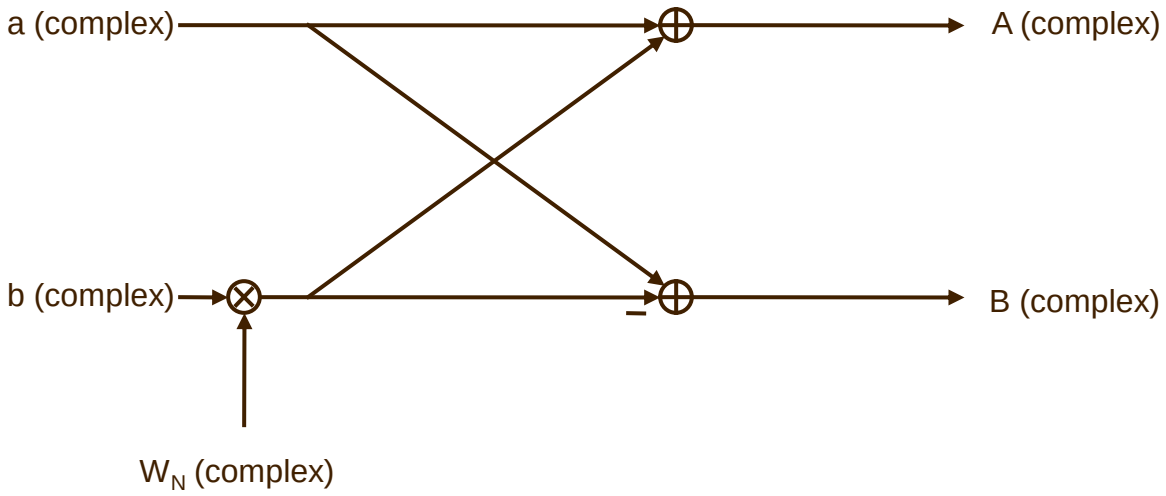
Decimation-in-Time (DIT) Radix-2 FFT

- Split x(n) into even and odd samples and perform smaller FFTs
  - $x_1(n) = x(2r)$
  - $x_2(n) = x(2r+1)$
  - n=0, 1, … N/2-1
- Radix-2 Decimation-in-Time (DIT) FFT algorithm
  - The “Butterfly" structure takes in 2 inputs and produces 2 outputs
  - Butterfly operation implements the basic 2-point FFT
- Computations:
  - $(N/2)\log_2 N$ Complex Multiplications
  - $N\log_2 N$ Complex Additions

Radix-2 DIT FFT Basic Butterfly Operation



Basic butterfly computation in the decimation-in-time FFT



Bit Reversal (16-point Sequence)

The input sequence needs to be applied in a strange order !

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	2	4	6	8	10	12	14	1	3	5	7	9	11	13	15
0000	0010	0100	0110	1000	1010	1100	1110	0001	0011	0101	0111	1001	1011	1101	1111
0	4	8	12	2	6	10	14	1	5	9	13	3	7	11	15
0000	0100	1000	1100	0010	0110	1010	1110	0001	0101	1001	1101	0011	0111	1011	1111
0	8	4	12	2	10	6	14	1	9	5	13	3	11	7	15
0000	1000	0100	1100	0010	1010	0110	1110	0001	1001	0101	1101	0011	1011	0111	1111

So abcd $\rightarrow$ bcda $\rightarrow$ cdba $\rightarrow$ dcba

The bits of the index have been reversed !

(DSP/FFT Processors have a Special Addressing Mode for this Purpose)

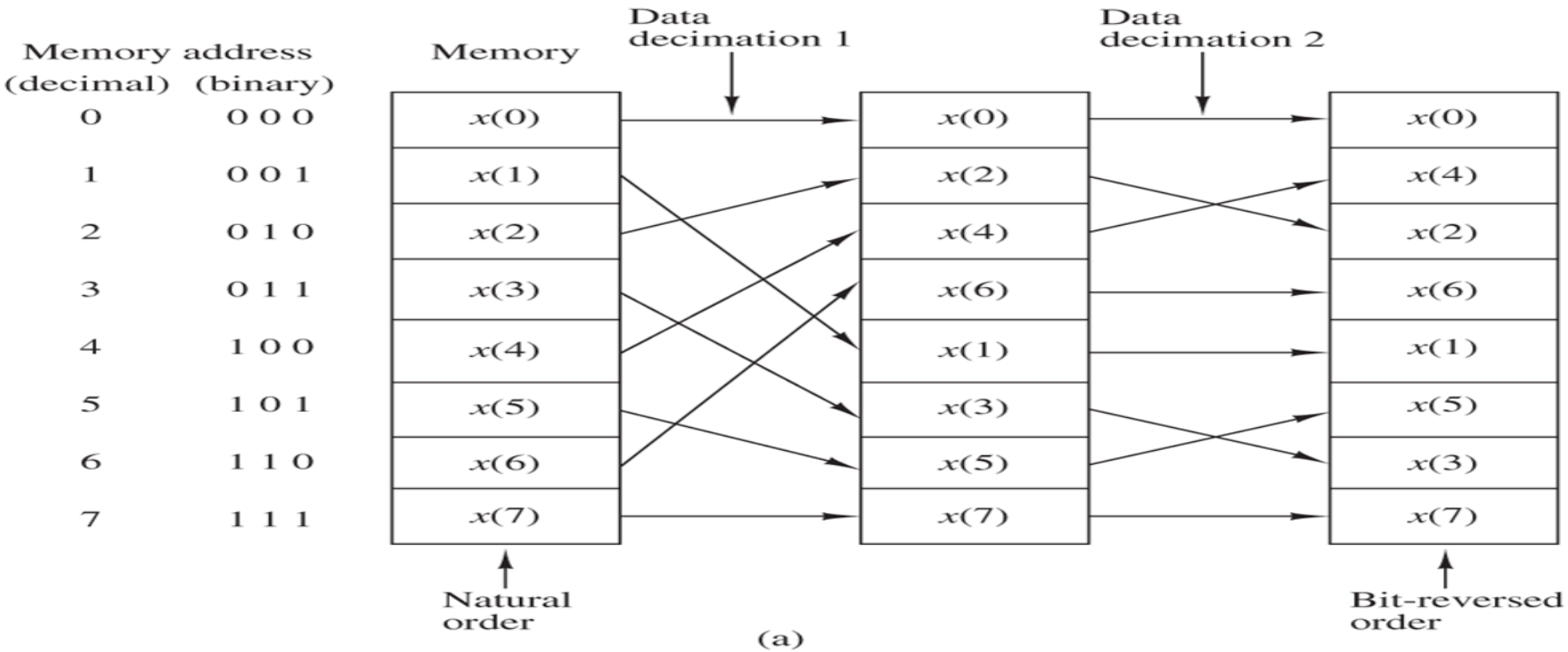
Bit Reversal (8-point Sequence)

The input sequence needs to be applied in a strange order !

0	1	2	3	4	5	6	7
000	001	010	011	100	101	110	111
0	2	4	6	1	3	5	7
000	010	100	110	001	011	101	111
0	4	2	6	1	5	3	7
000	100	010	110	001	101	011	111

The bits of the index have been reversed !

Bit-Reversing for 8-point Radix-2 DIT FFT



$(n_2 n_1 n_0)$	→	$(n_1 n_0 n_2)$	→	$(n_0 n_1 n_2)$
(0 0 0)	→	(0 0 0)	→	(0 0 0)
(0 0 1)	→	(0 1 0)	→	(1 0 0)
(0 1 0)	→	(1 0 0)	→	(0 1 0)
(0 1 1)	→	(1 1 0)	→	(1 1 0)
(1 0 0)	→	(0 0 1)	→	(0 0 1)
(1 0 1)	→	(0 1 1)	→	(1 0 1)
(1 1 0)	→	(1 0 1)	→	(0 1 1)
(1 1 1)	→	(1 1 1)	→	(1 1 1)

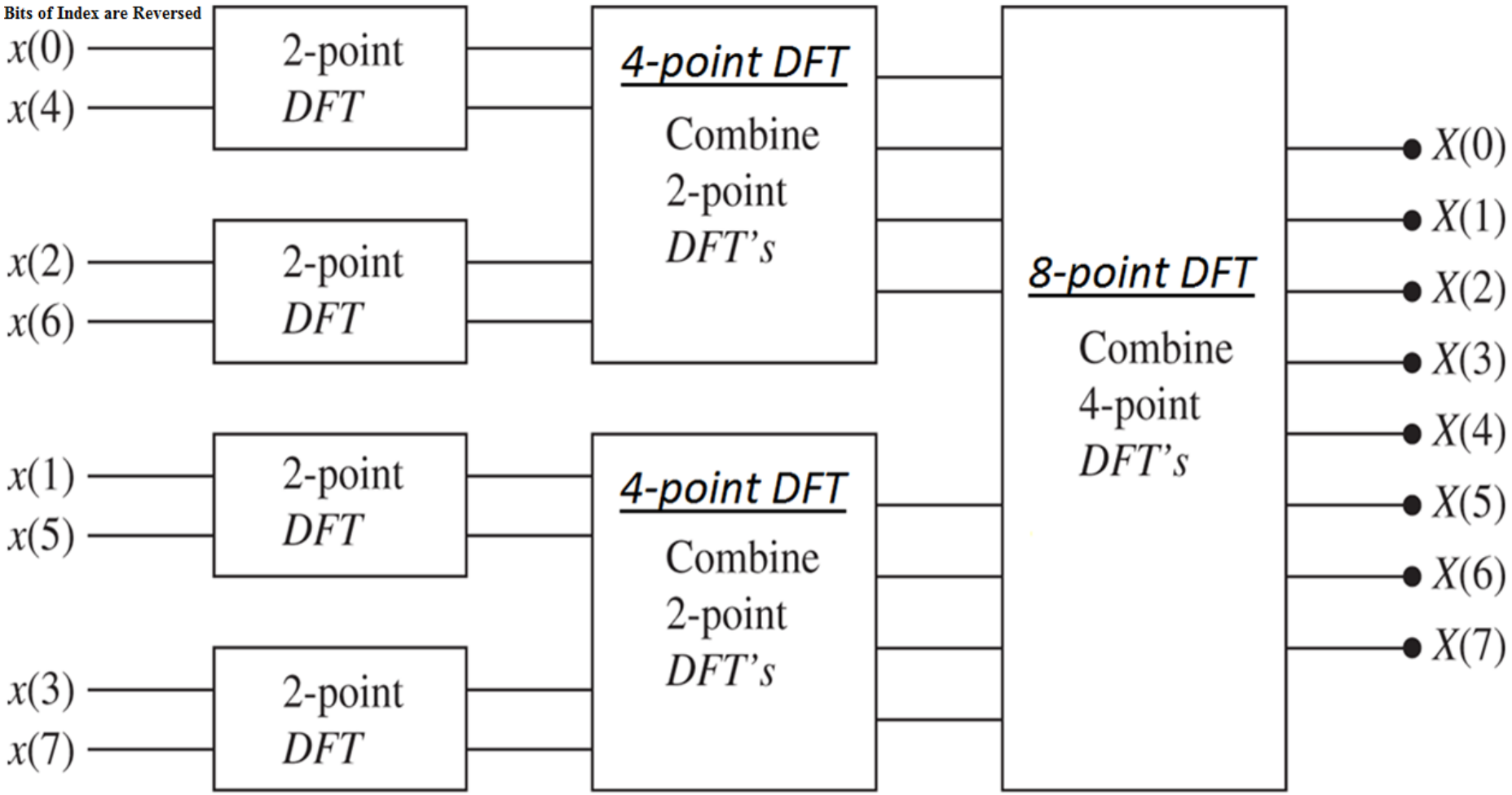
(b)

Shuffling of the data and bit reversal.

Note on Bit-Reversed Order

- For Radix-2 Decimation-in-Time FFT
  - If $x(n)$ is in bit-reversed order (n=0,4,2,6,1,5,3,7), then output $X(k)$ is in natural order (k=0,1,2,3,4,5,6,7)
  - Vice-versa also holds true
- For Radix-2 Decimation-in-Frequency FFT
  - If $x(n)$ is in natural order (n=0,1,2,3,4,5,6,7), then output $X(k)$ is in bit-reversed order (k=0,4,2,6,1,5,3,7)
  - Vice-versa also holds true
- Bit-Reversing is an inherent property of FFTs

8-point Radix-2 DIT FFT Computation



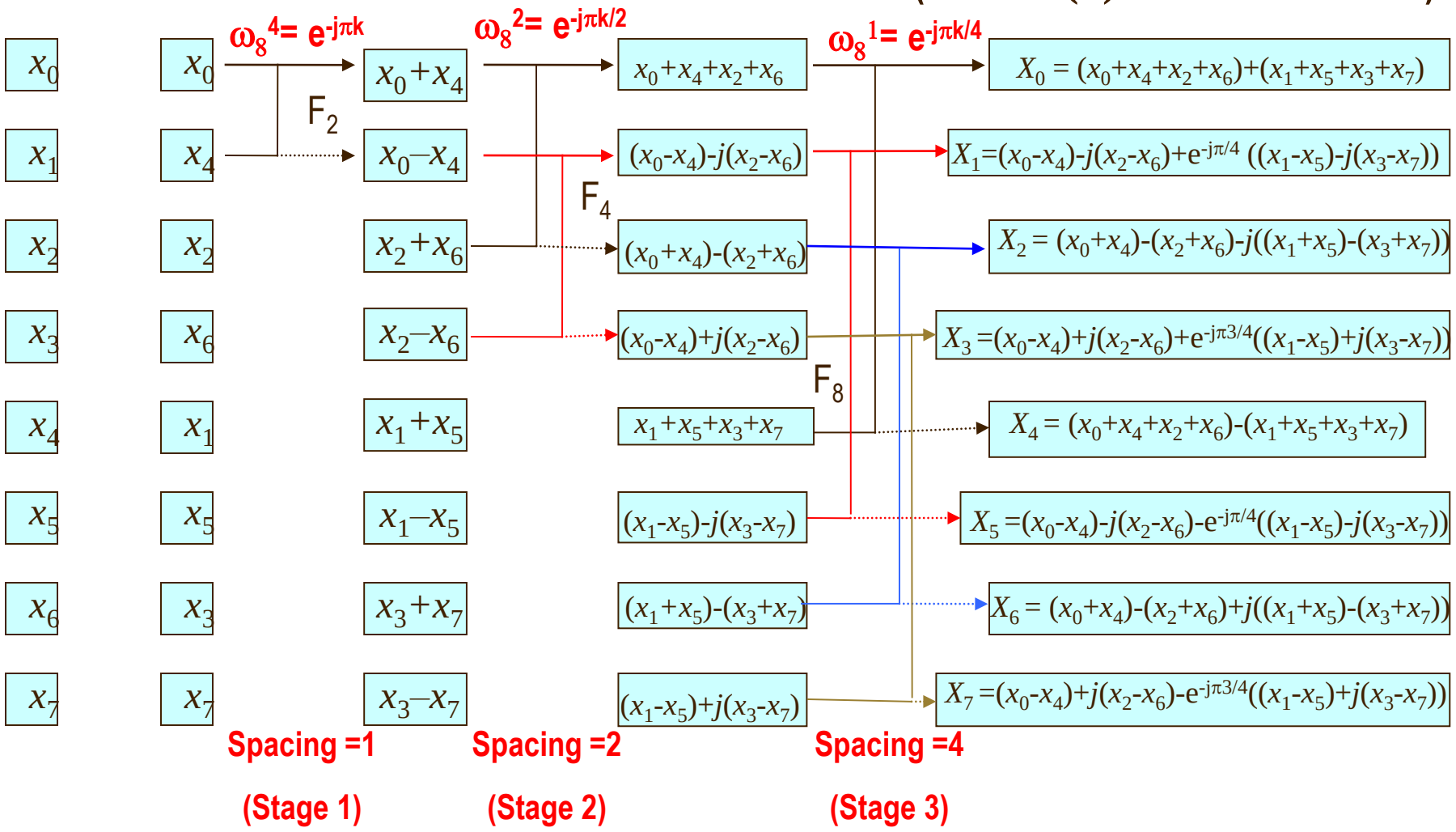
Three stages in the computation of an $N = 8$-point DFT.

8-point Radix-2 DIT FFT Computation

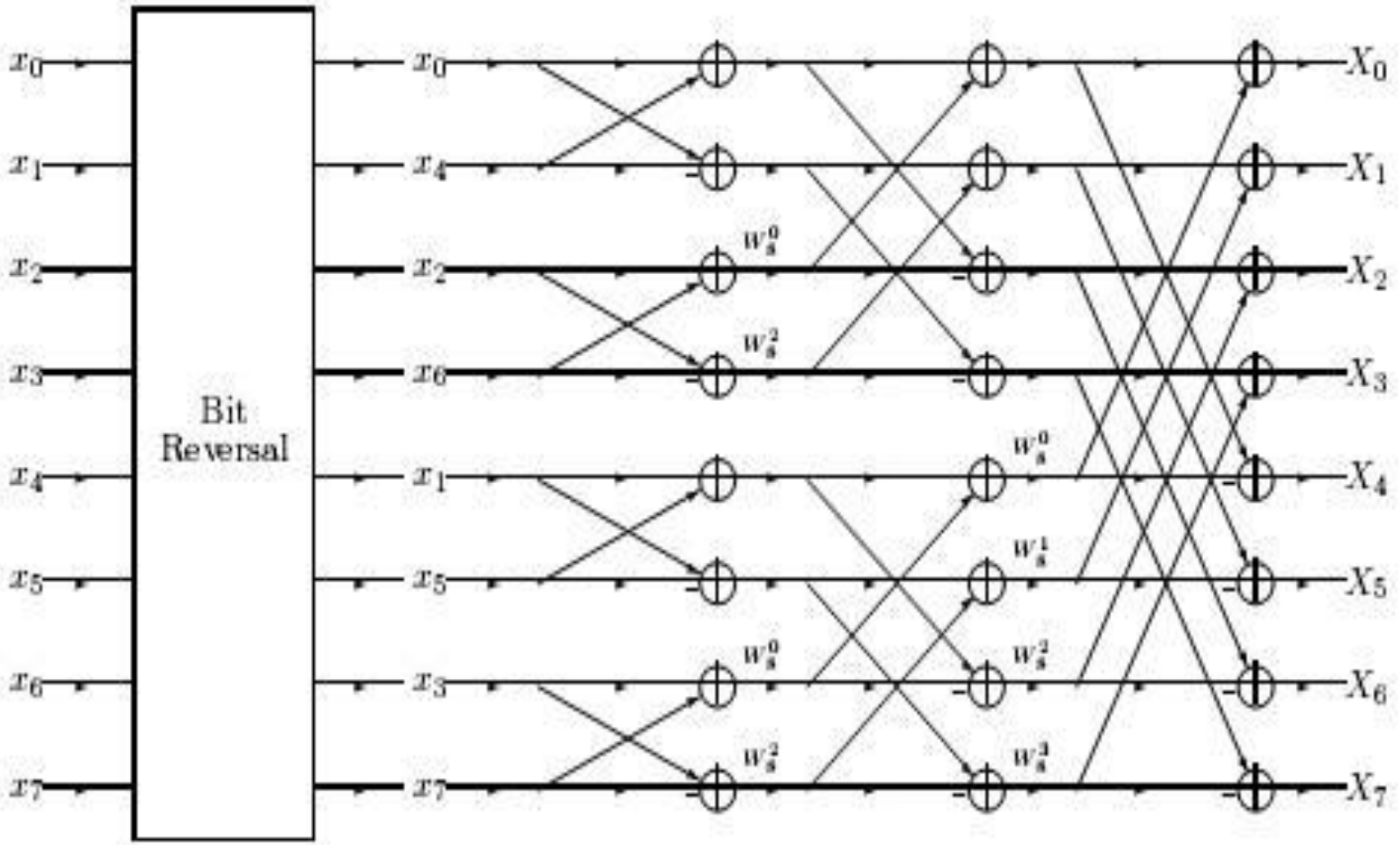
Swap Data according to the Bit Reversal Logic

Note: In EE,ECE,AI/CSE/IT we use *j* instead of *i*

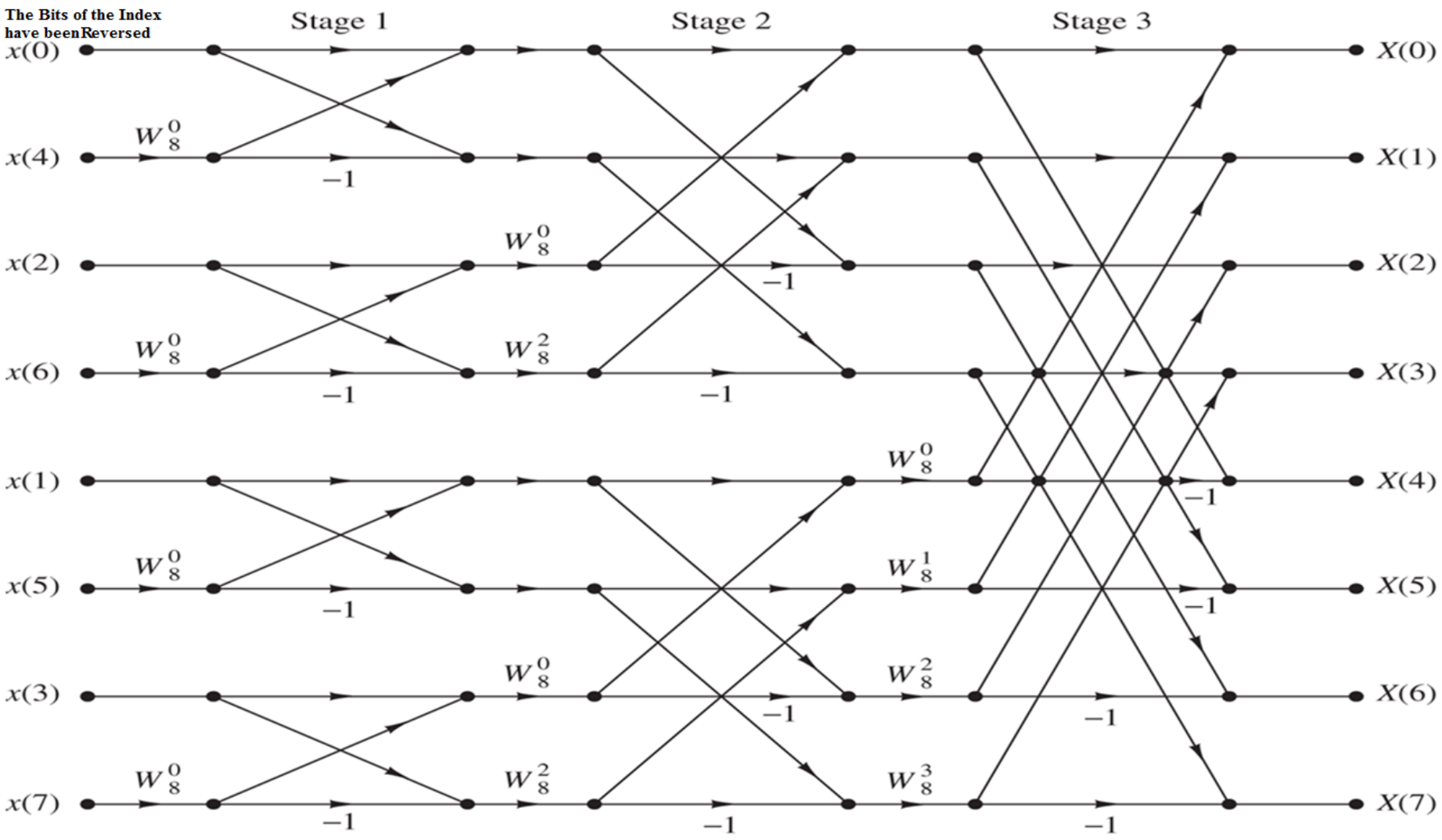
In Place FFT Computation (FFT of $x(n)$ is in natural order)



8 Point Radix-2 DIT FFT Signal-Flow Graph



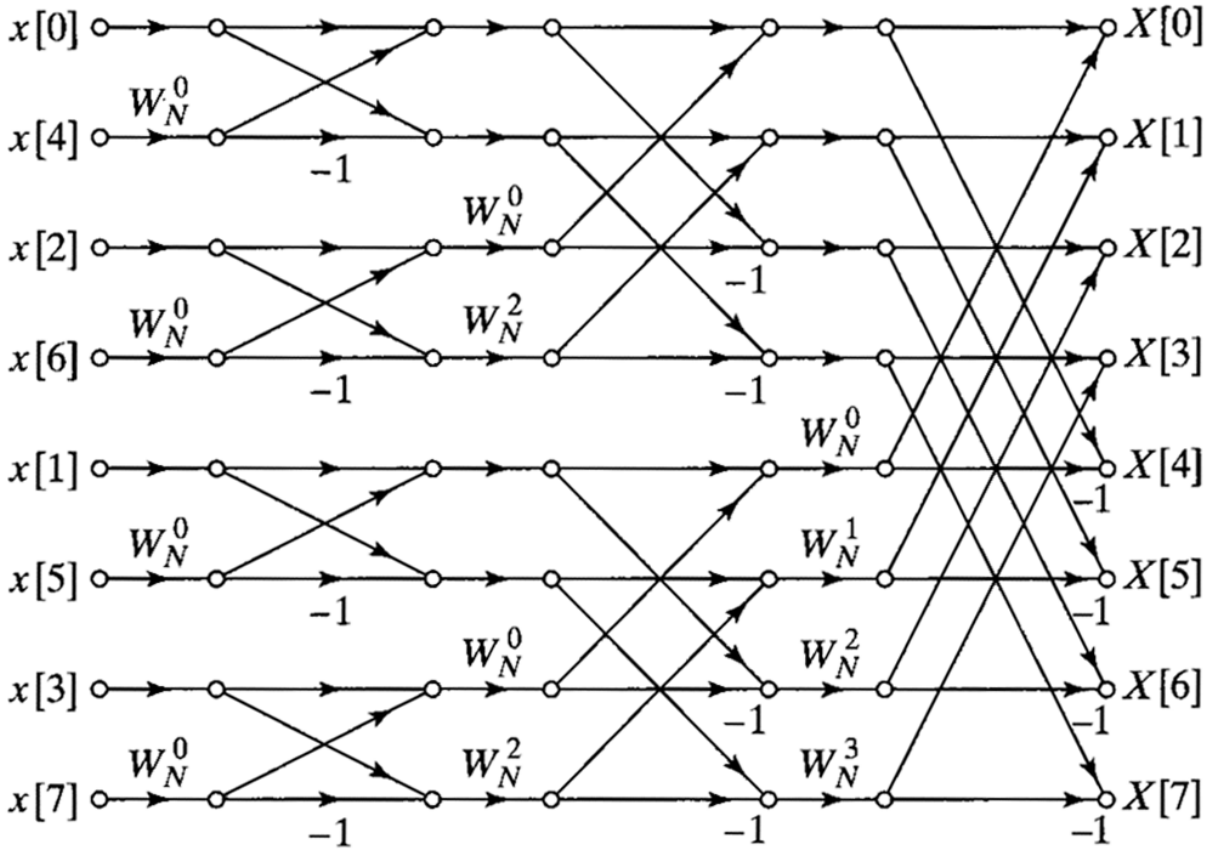
8-point Radix-2 DIT FFT Signal-Flow Graph



Eight-point decimation-in-time FFT algorithm.

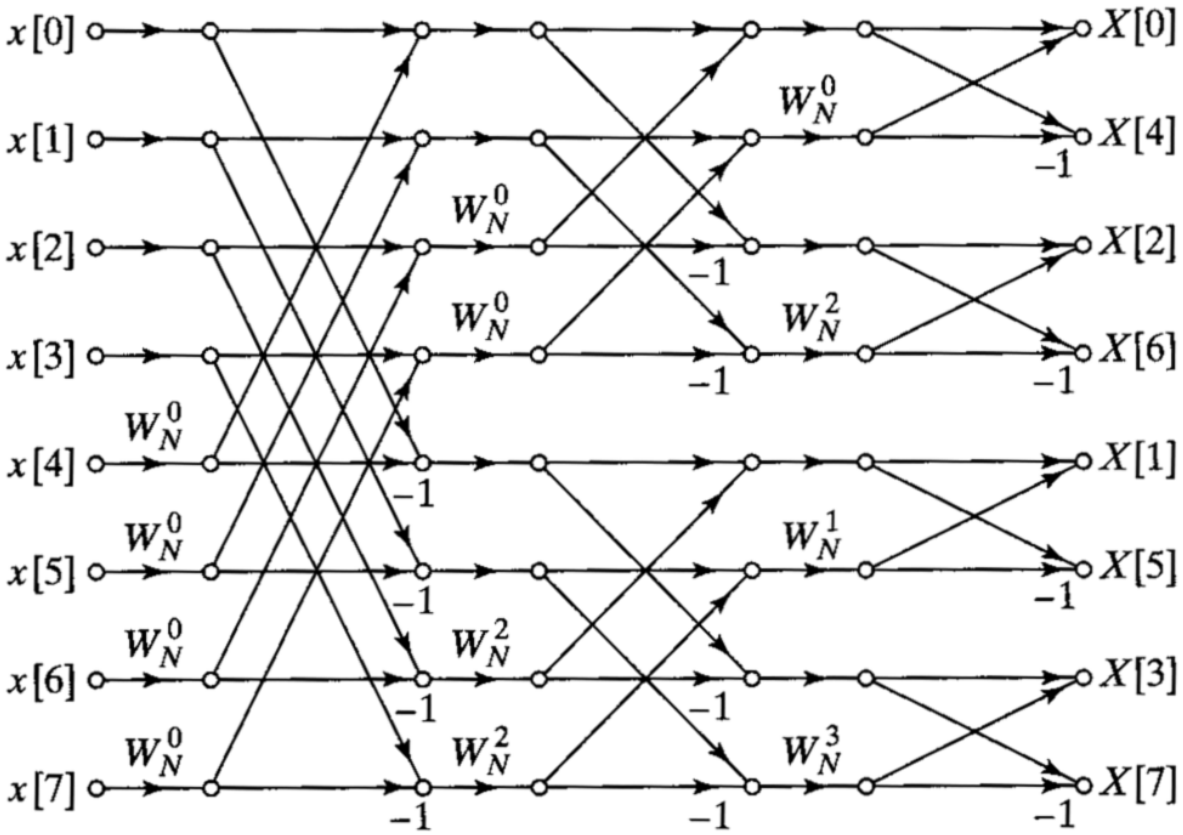
Alternate Structures for Radix 2 DIT FFT

- DIT structure with input bit-reversed, output natural:



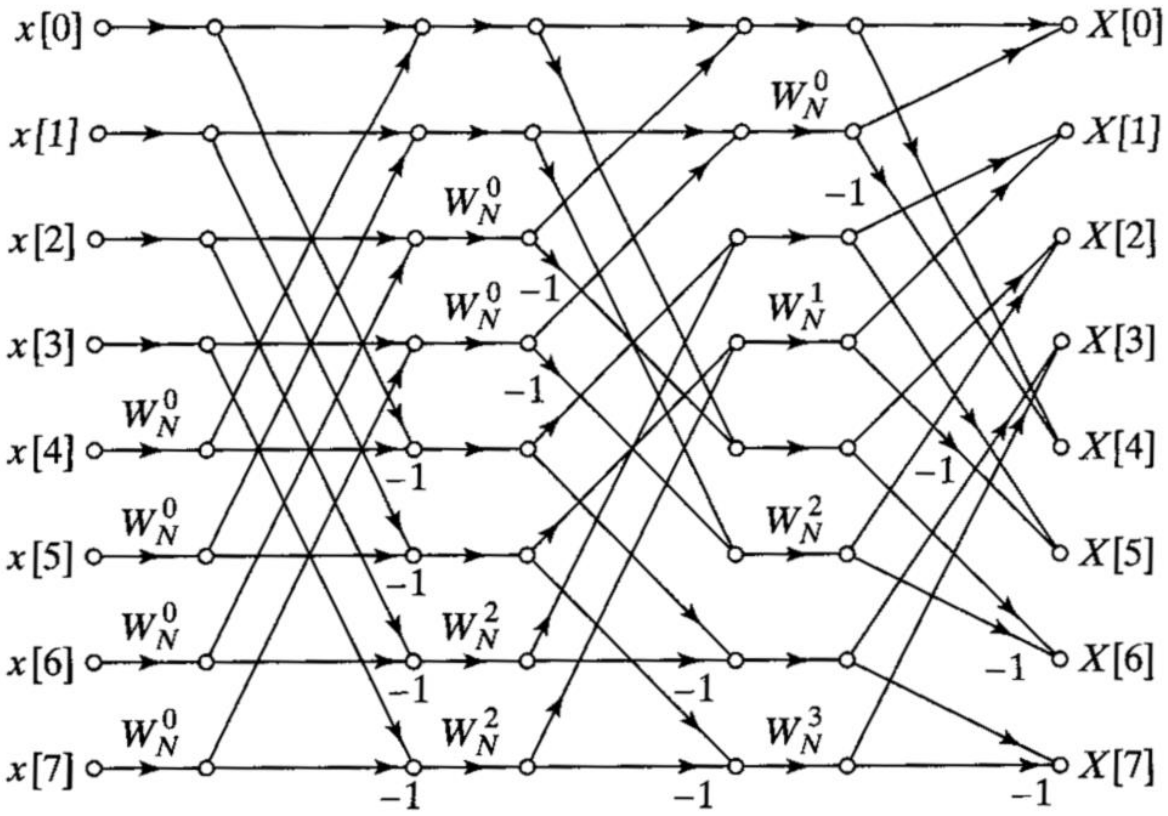
Alternate DIT FFT Structures (continued)

- DIT structure with input natural, output bit-reversed:



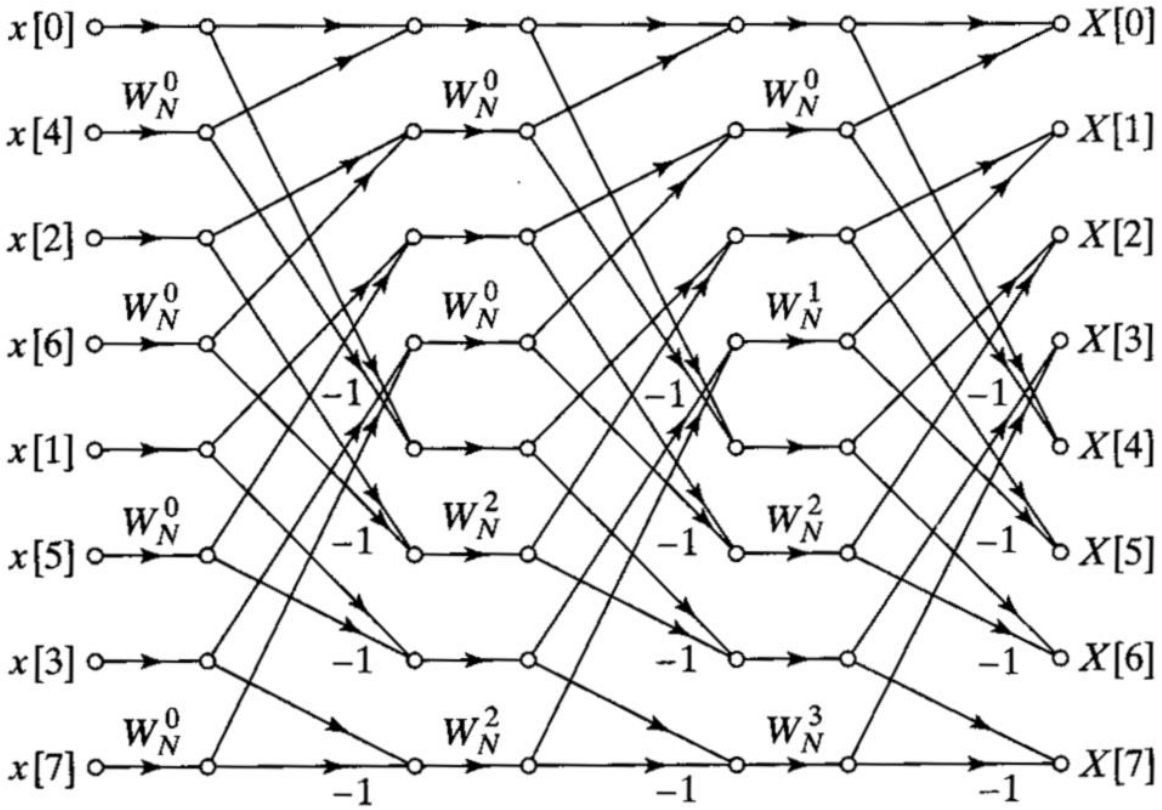
Alternate DIT FFT Structures (continued)

- DIT structure with both input and output natural:



Alternate DIT FFT Structures (continued)

- DIT structure with same structure for each stage:



Comments on Alternate FFT Structures

- A method to avoid bit-reversal in filtering operations is:
  - Compute the forward transform (DFT) using a natural input, bit-reversed output
  - Multiply the DFT coefficients of input and the filter response (both are in bit-reversed order)
  - Compute the inverse transform (IDFT) of the product using bit-reversed input and the natural output

Using FFTs for Inverse DFTs

- We've always been talking about forward DFTs in our discussion about FFTs …. what about the inverse FFT?

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]W_N^{-kn}; \quad X[k] = \sum_{n=0}^{N-1} x[n]W_N^{kn}$$

- One way to modify the FFT algorithm for the computation of an inverse DFT (IDFT or IFFT) is:
  - Replace W_N^k by W_N^{-k} wherever it appears
  - Multiply final output by $1/N$
- This method has the disadvantage that it requires modifying the internal code in the FFT subroutine

A Better Way to Modify FFT Code for Inverse DFT

- Taking the Complex Conjugate of both sides of the IDFT (IFFT) equation and multiplying by *N*:

$$Nx^*[n] = \frac{1}{N}\sum_{k=0}^{N-1} X^*[k]W_N^{kn}; \text{ or } x[n] = \frac{1}{N}\left[\sum_{k=0}^{N-1} X^*[k]W_N^{kn}\right]^*$$

- This suggests that we can modify the FFT algorithm for the inverse DFT (FFT) computation by the following:
  - Complex Conjugate the Input DFT Coefficients
  - Compute the *Forward* FFT
  - Complex Conjugate the Output of the FFT and Multiply by $1/N$
- This method has the advantage that the internal FFT code is undisturbed; hence it is widely used.

Decimation-in-Frequency Radix-2 FFT

- Decompose X(k) such that it is split into FFT of points 0 to N/2-1 and points N/2 to N-1
  - Then decimate X(k) into even and odd numbered samples
- Radix-2 Decimation-in-Frequency (DIF) FFT algorithm
  - The “Butterfly" operation needs 2 inputs to give 2 outputs
  - Butterfly operation implements 2-point FFT
- Computations:
  - $(N/2).\log_2 N$ Complex Multiplications
  - $N.\log_2 N$ Complex Additions

Decimation-in-Frequency Radix-2 FFT

- Consider the original DFT equation ….

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk}$$

- Separate the 1st half and the 2nd half of time samples:

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

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

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

- Note that these are not *N*/2-point DFTs

Decimation-in-Frequency Radix-2 FFT

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

- For *k* even, let $k=2r$

$$X[k]=\sum_{n=0}^{(N/2)-1}\left[x[n]+(-1)^{2r} x[n+(N/2)]\right]W_N^{n2r}=\sum_{n=0}^{(N/2)-1}[x[n]+x[n+(N/2)]]W_{N/2}^{nr}$$

- For *k* odd, let $k=2r+1$

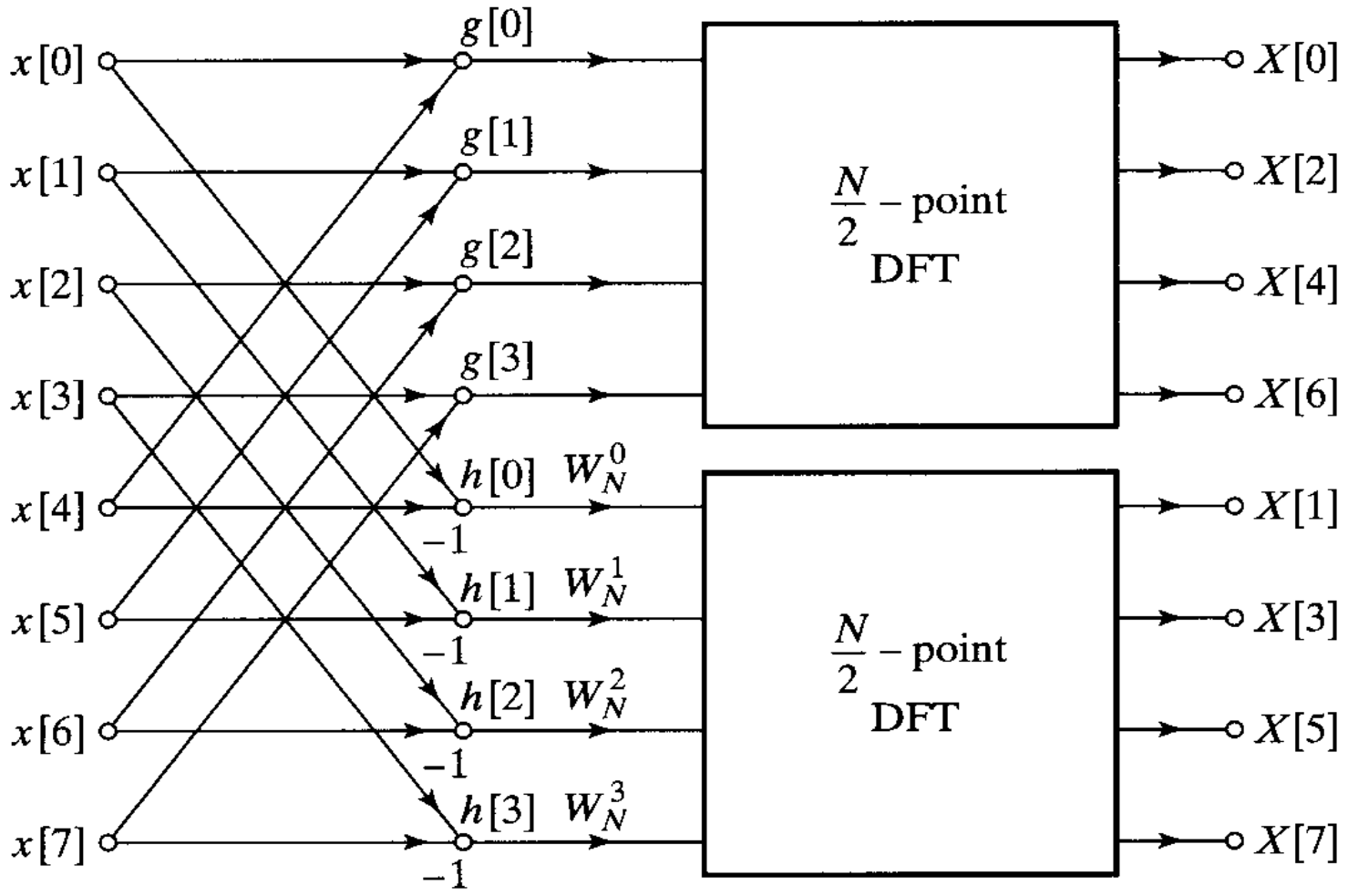
$$X[k]=\sum_{n=0}^{(N/2)-1}\left[x[n]+(-1)^{2r}(-1)x[n+(N/2)]\right]W_N^{n(2r+1)}$$

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

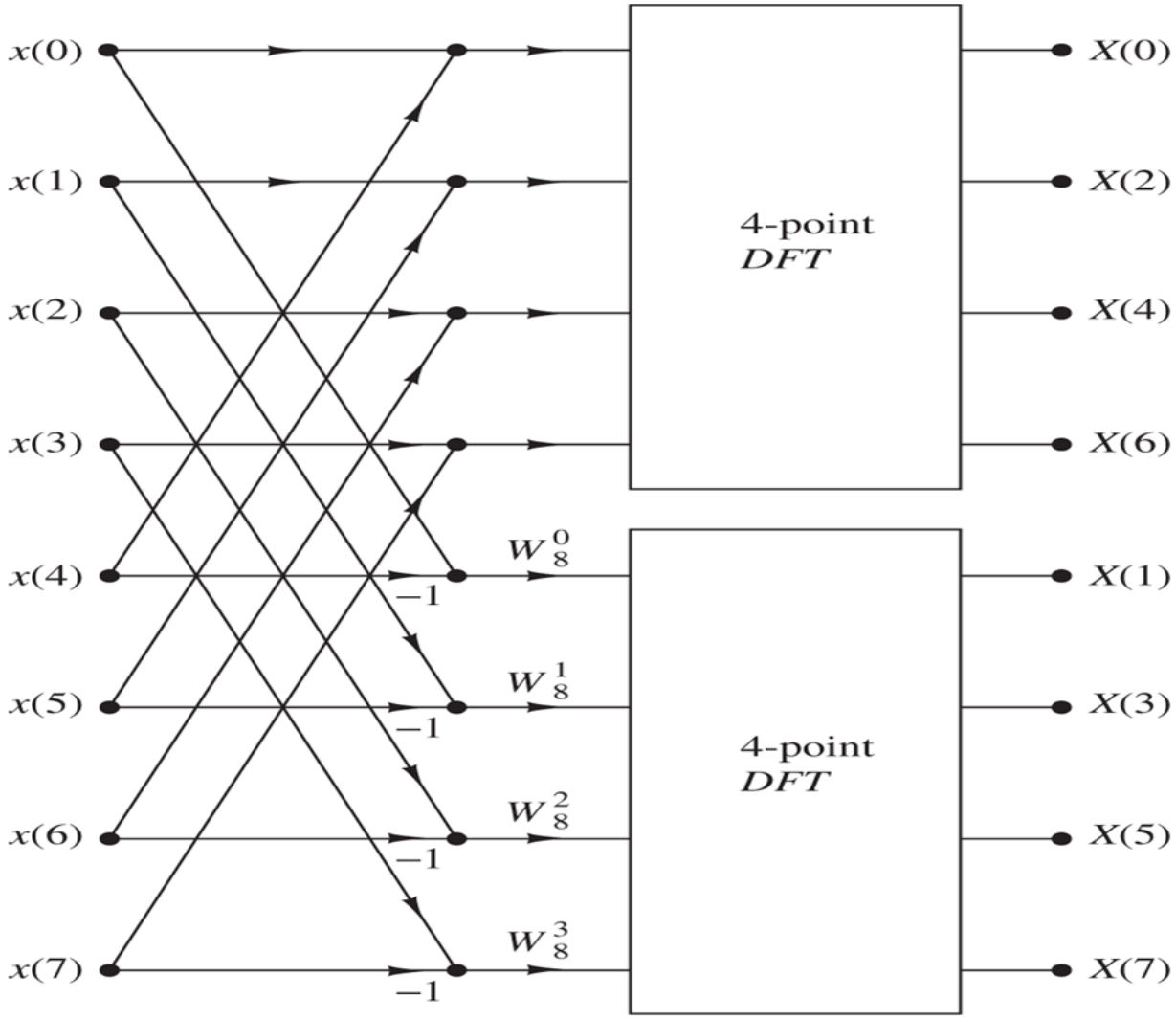
- These expressions are the *N*/2-point DFTs of

$$x[n]+x[n+(N/2)] \text{ and } [x[n]-x[n+(N/2)]]W_N^n$$

Decimation-in-Frequency Radix-2 FFT (N=8)

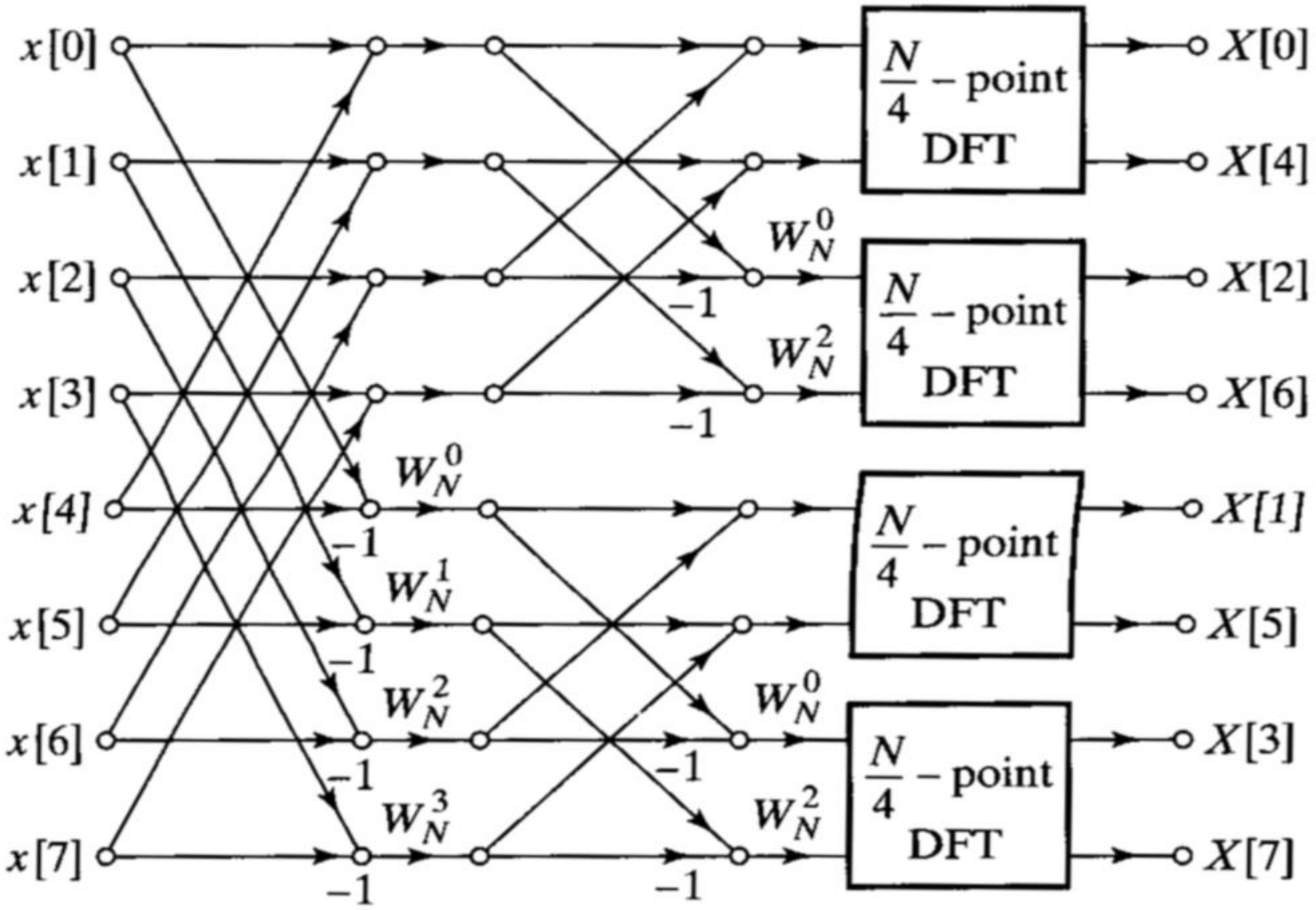


Decimation-in-Frequency Radix-2 FFT (N=8)

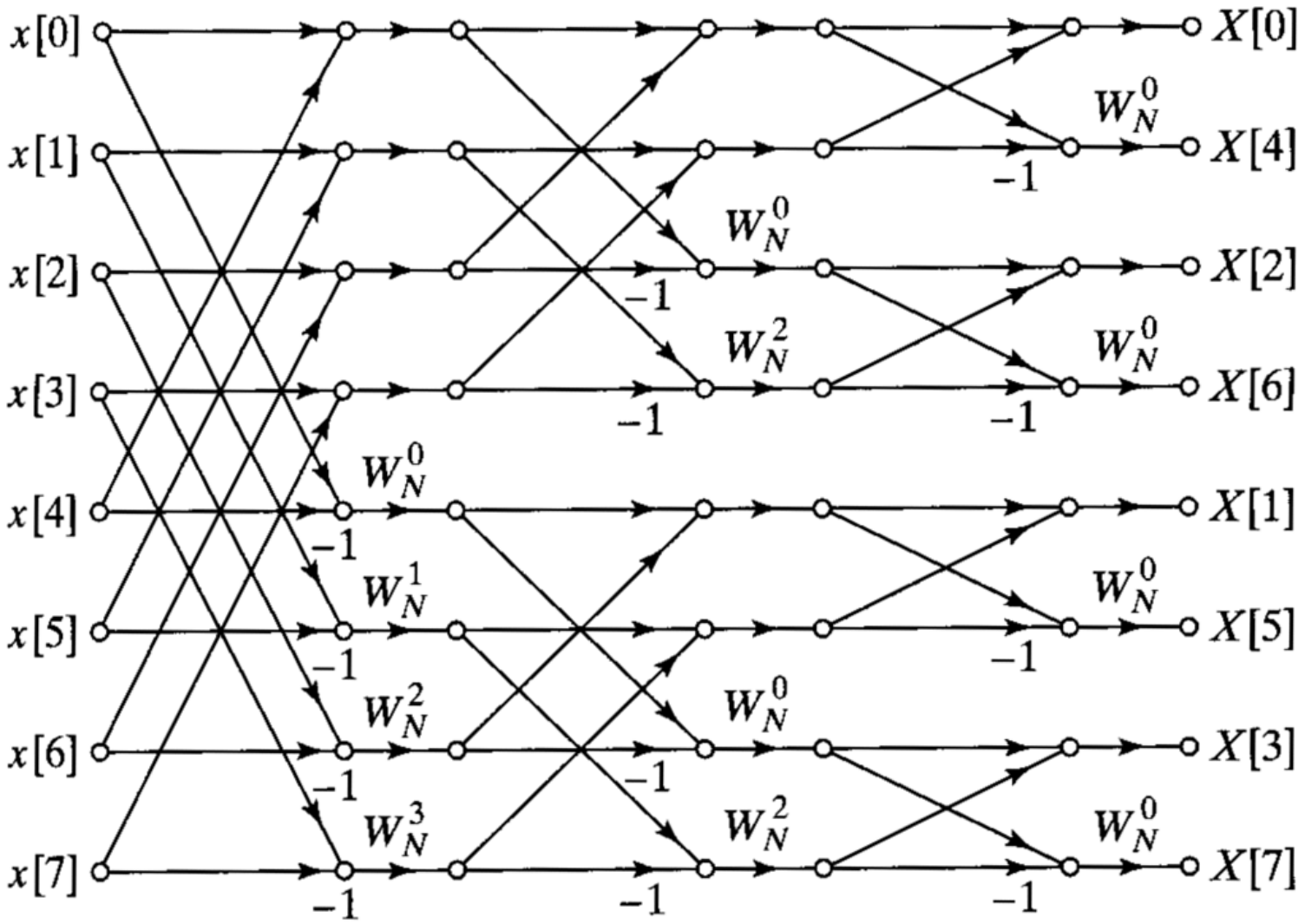


First stage of the decimation-in-frequency FFT algorithm.

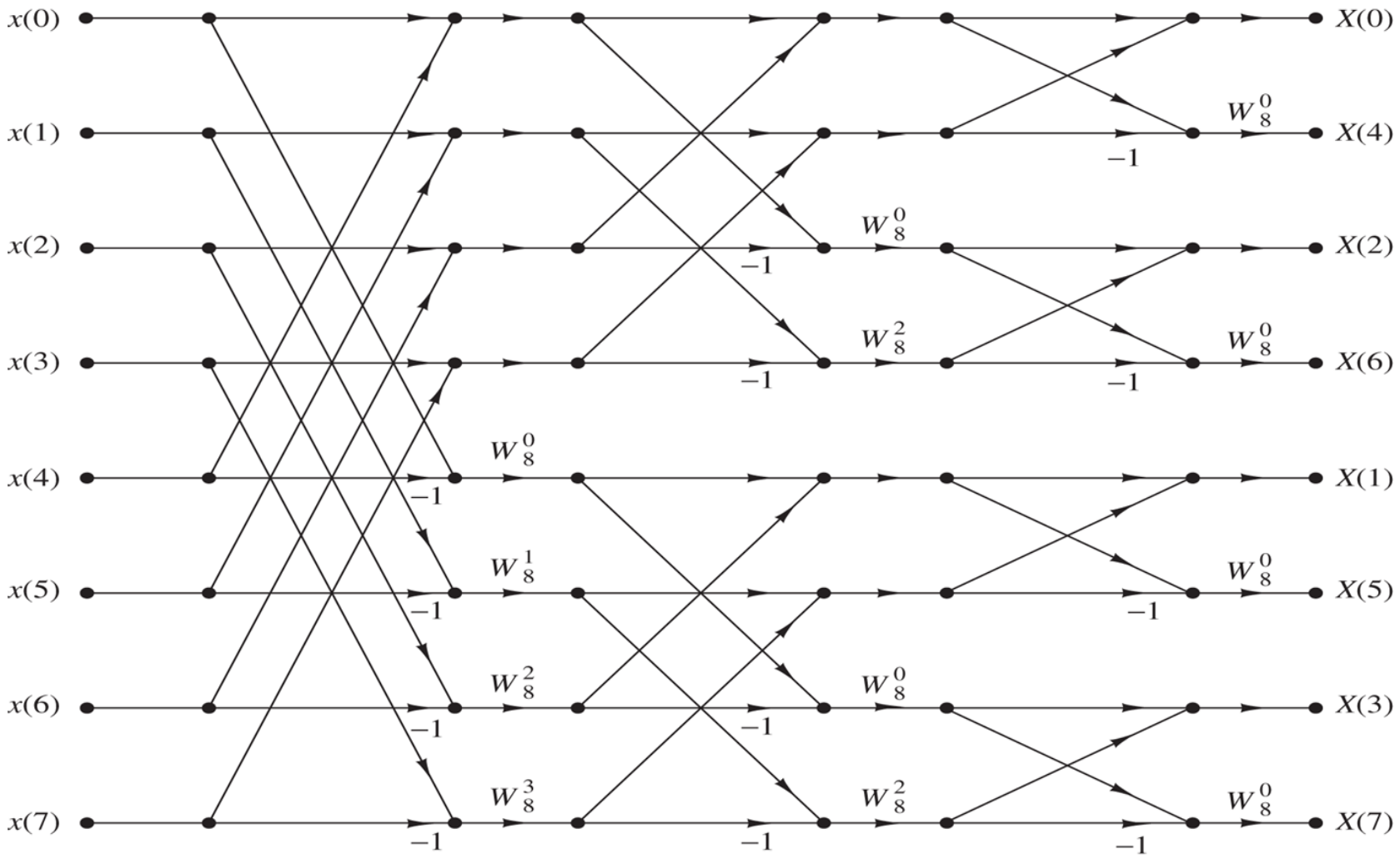
Continuing by decomposing the odd and even *output* points we obtain …



… and replacing the *N*/4-point DFTs by butterflies we obtain

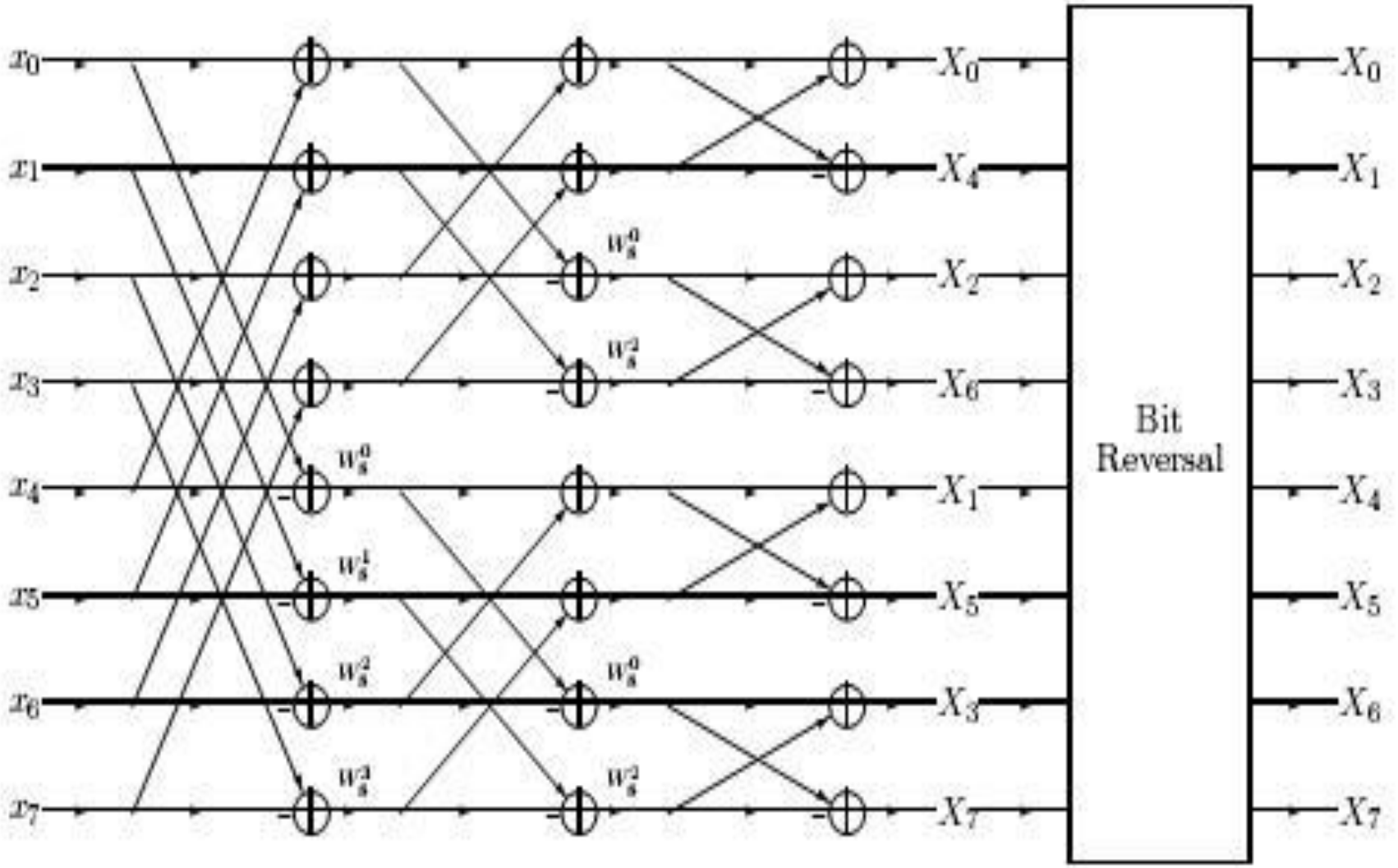


Decimation-in-Frequency Radix-2 FFT (N=8)

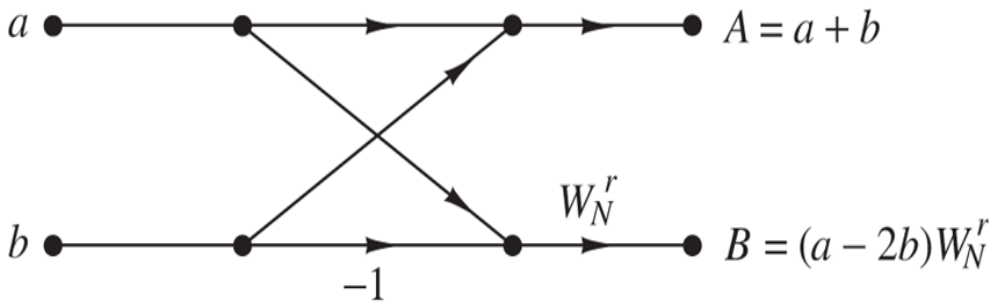


$N = 8$-point decimation-in-frequency FFT algorithm.

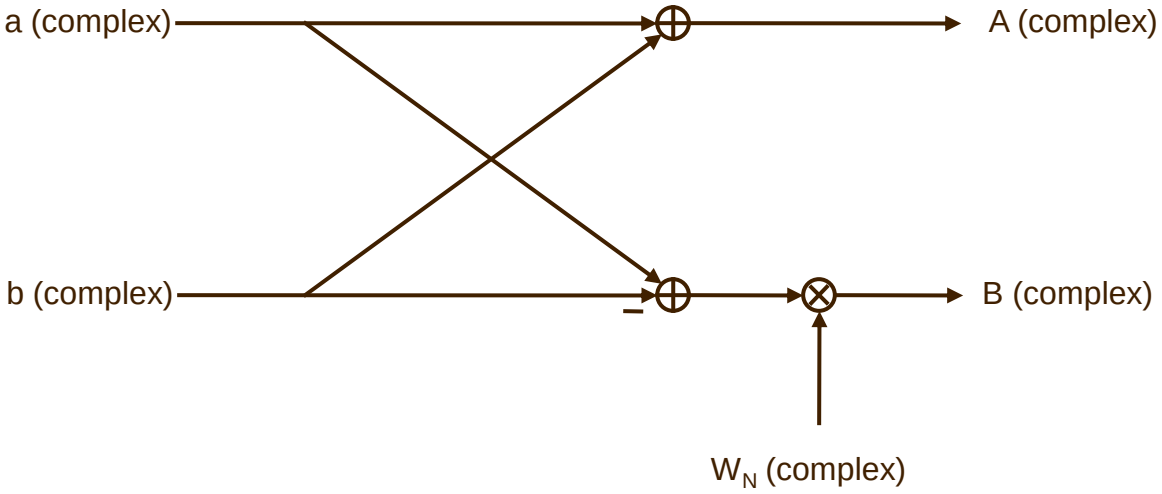
8-point Radix-2 DIF FFT Signal-flow Graph



Radix-2 DIF Basic Butterfly

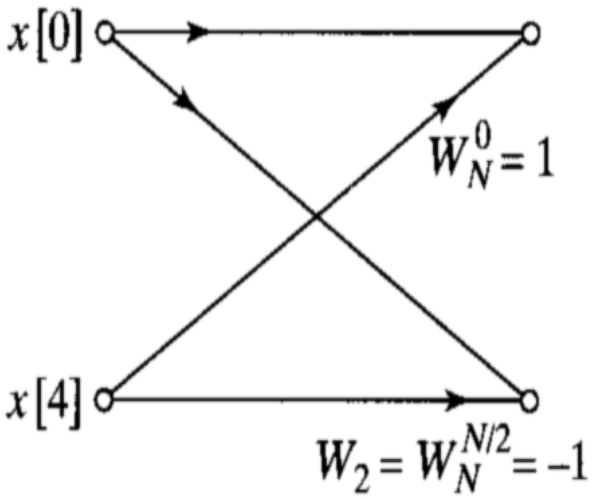


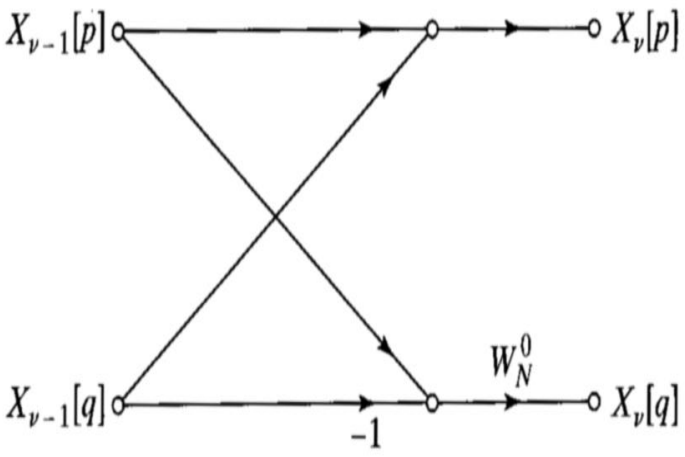
Basic butterfly computation in the decimation-in-frequency FFT algorithm.



The DIF FFT is the *Transpose* of the DIT FFT

- To obtain the flowgraph transpose:
  - Reverse direction of flowgraph arrows
  - Interchange input(s) and output(s)
- DIT Butterfly: DIF Butterfly:

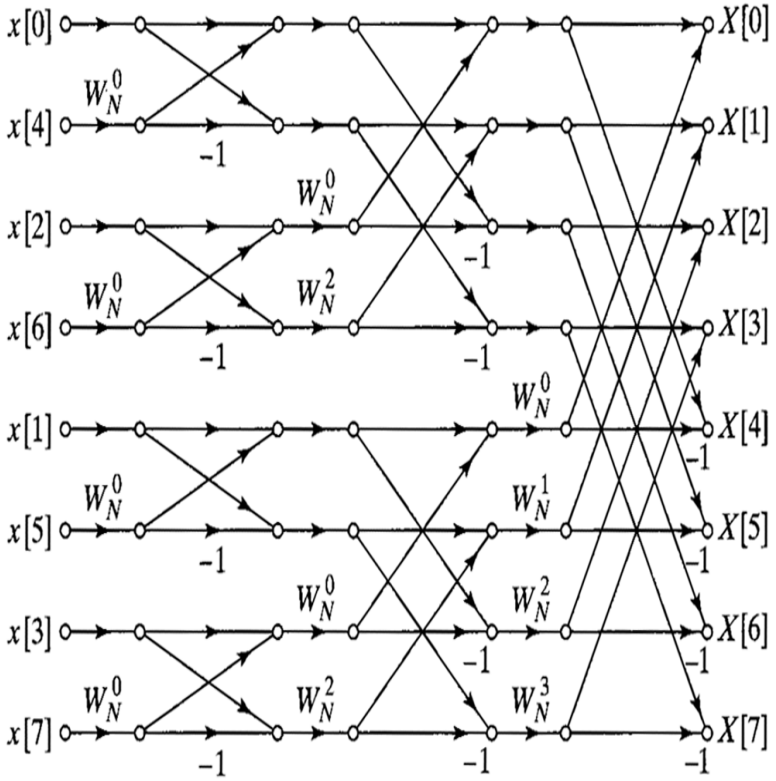




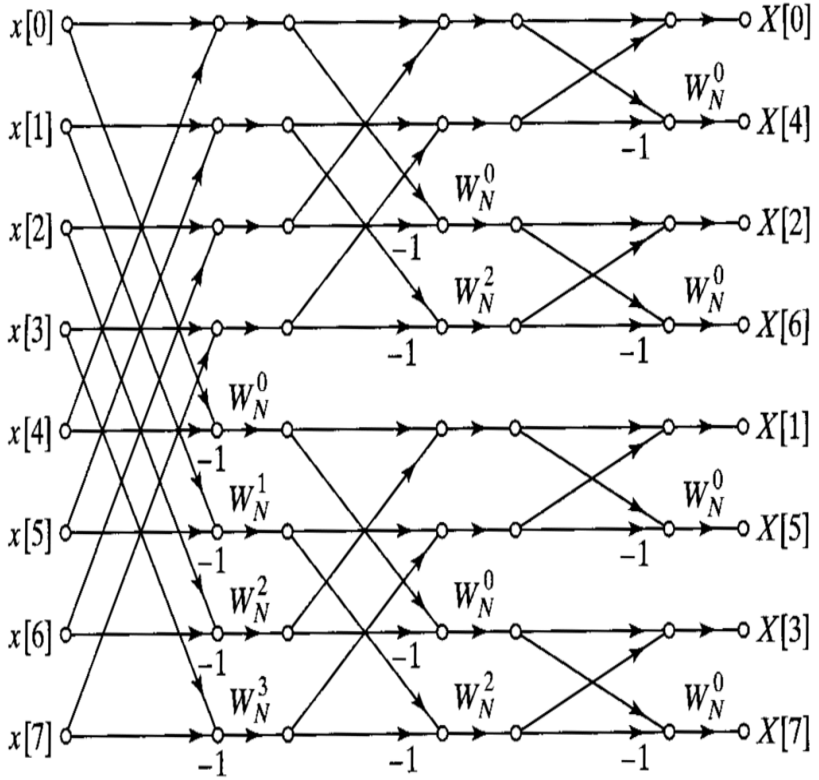
The DIF FFT is the *Transpose* of the DIT FFT

- Comparing DIT and DIF Structures:

DIT FFT Structure:



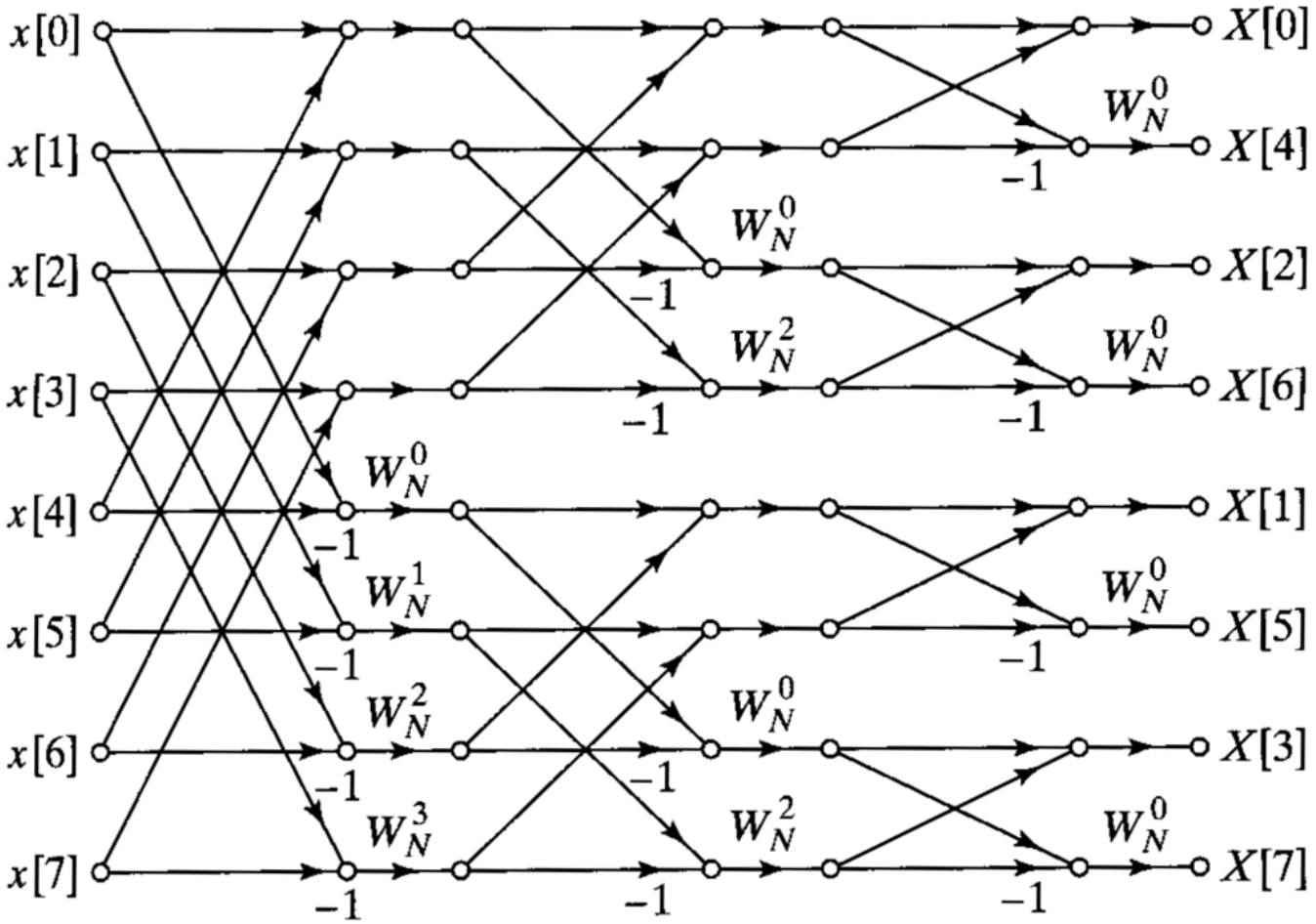
DIF FFT Structure:



- **Alternate forms for DIF FFTs are similar to those of DIT FFTs**

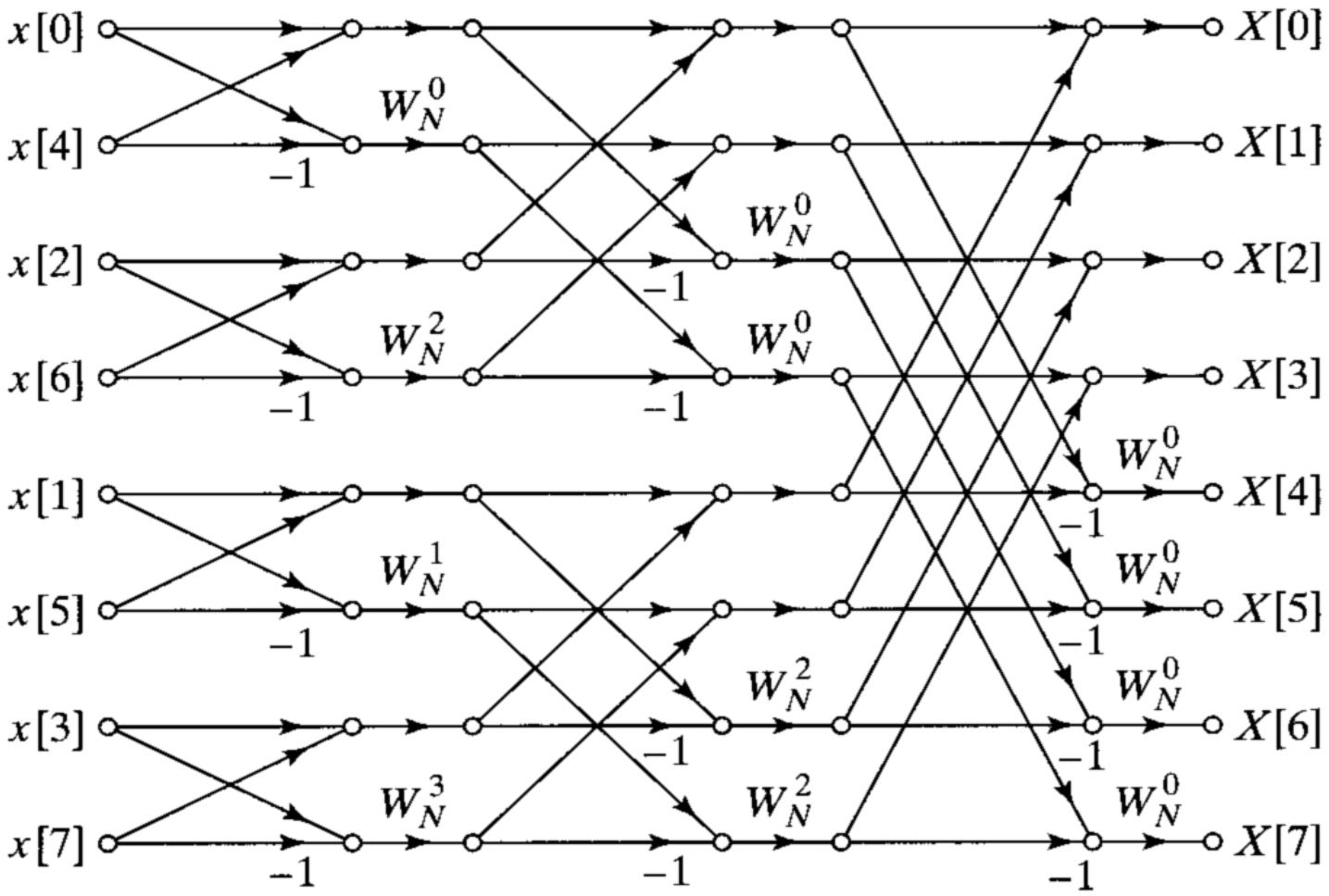
Alternate DIF FFT Structures

- DIF Structure with Natural Input, Bit-reversed Output:



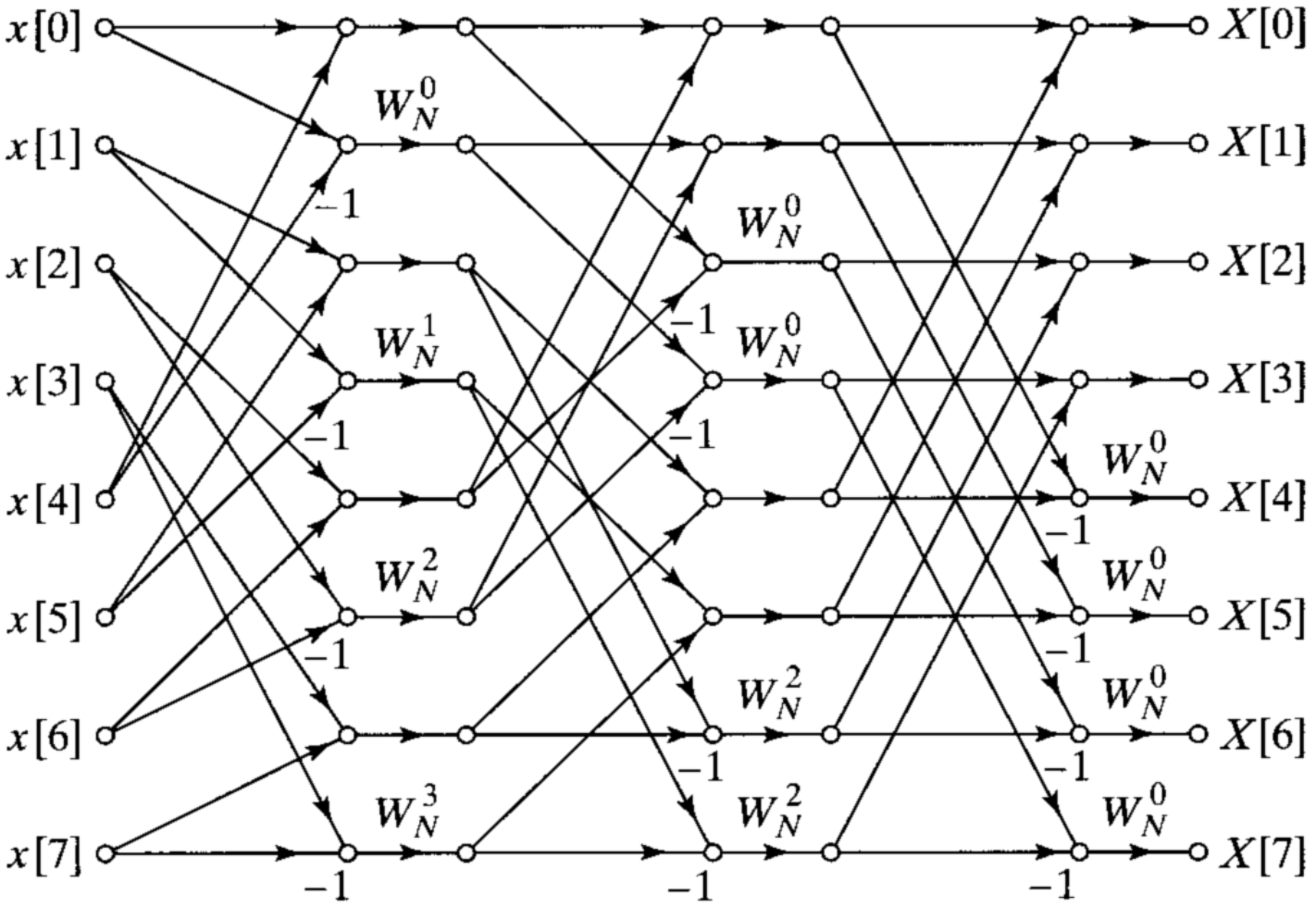
Alternate DIF FFT Structures (continued)

- DIF Structure with Bit-reversed Input, Natural Output:



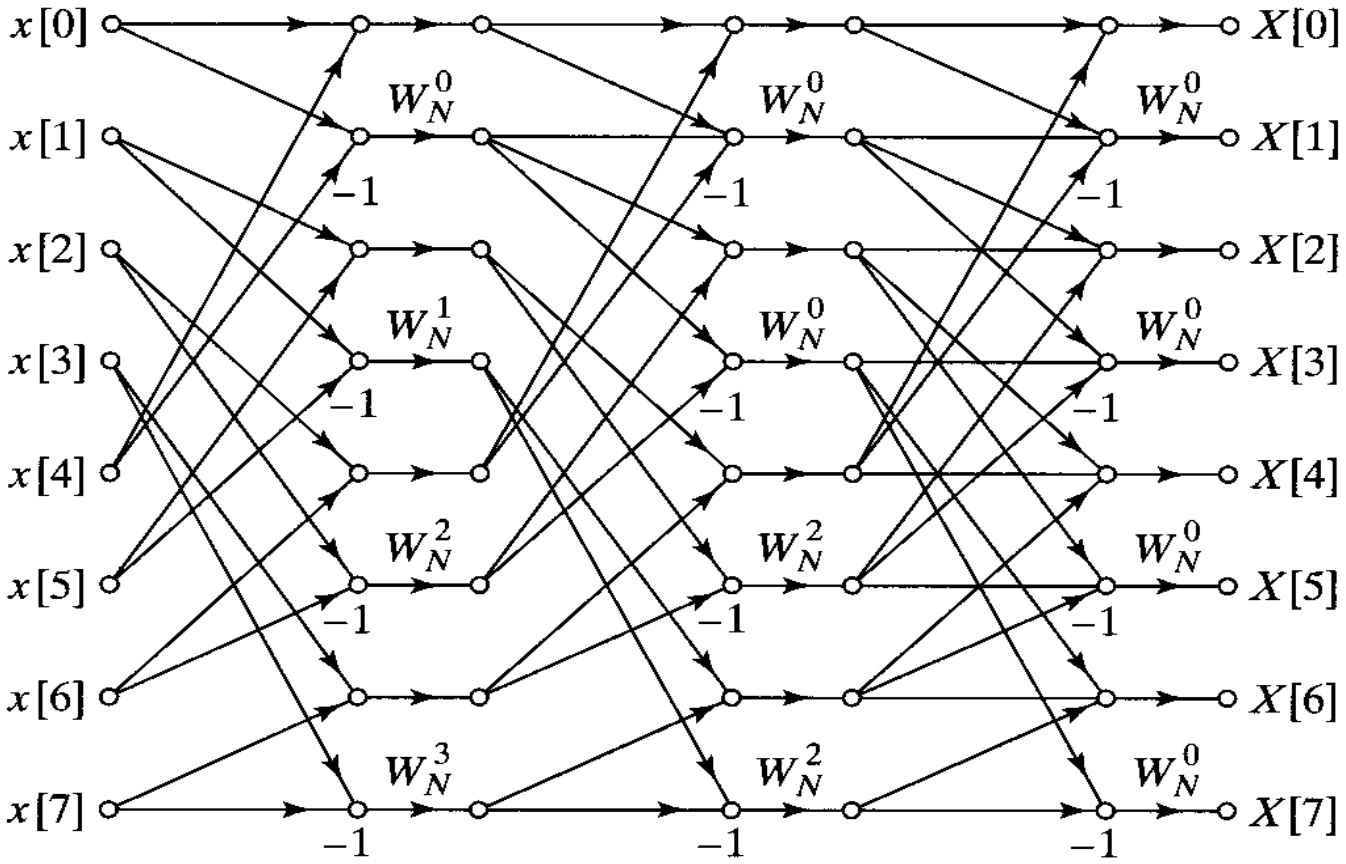
Alternate DIF FFT Structures (continued)

- DIF Structure with Natural Input and Natural Output



Alternate DIF FFT Structures (continued)

- DIF Structure with Identical Structure for Each Stage:



FFT Structures for Other DFT Sizes

- Can we do anything when the DFT size *N* is not an integer power of 2 (the non-radix 2 case)?
- Yes! Consider a value of *N* that is not a power of 2, but that still is highly factorable …

$$\text{Let } N = p_1 p_2 p_3 p_4 \ldots p_\nu; \quad q_1 = N/p_1, \quad q_2 = N/p_1 p_2, \text{etc.}$$

- Then let

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk}$$

$$= \sum_{r=0}^{q_1-1} x[p_1 r] W_N^{p_1 rk} + \sum_{r=0}^{q_1-1} x[p_1 r+1] W_N^{(p_1 r+1)k} + \sum_{r=0}^{q_1-1} x[p_1 r+2] W_N^{(p_1 r+2)k} + \ldots$$

Non-Radix 2 FFTs (continued)

- An arbitrary term of the sum on the previous panel is

$$\sum_{r=0}^{q_1-1} x[p_1 r + l] W_N^{(p_1 r + l)k}$$

$$= \sum_{r=0}^{q_1-1} x[p_1 r + l] W_N^{p_1 rk} W_N^{lk} = W_N^{lk} \sum_{r=0}^{q_1-1} x[p_1 r + l] W_{q_1}^{rk}$$

- This is a DFT of size q_1 of points spaced by p_1

Non-Radix 2 FFTs (continued)

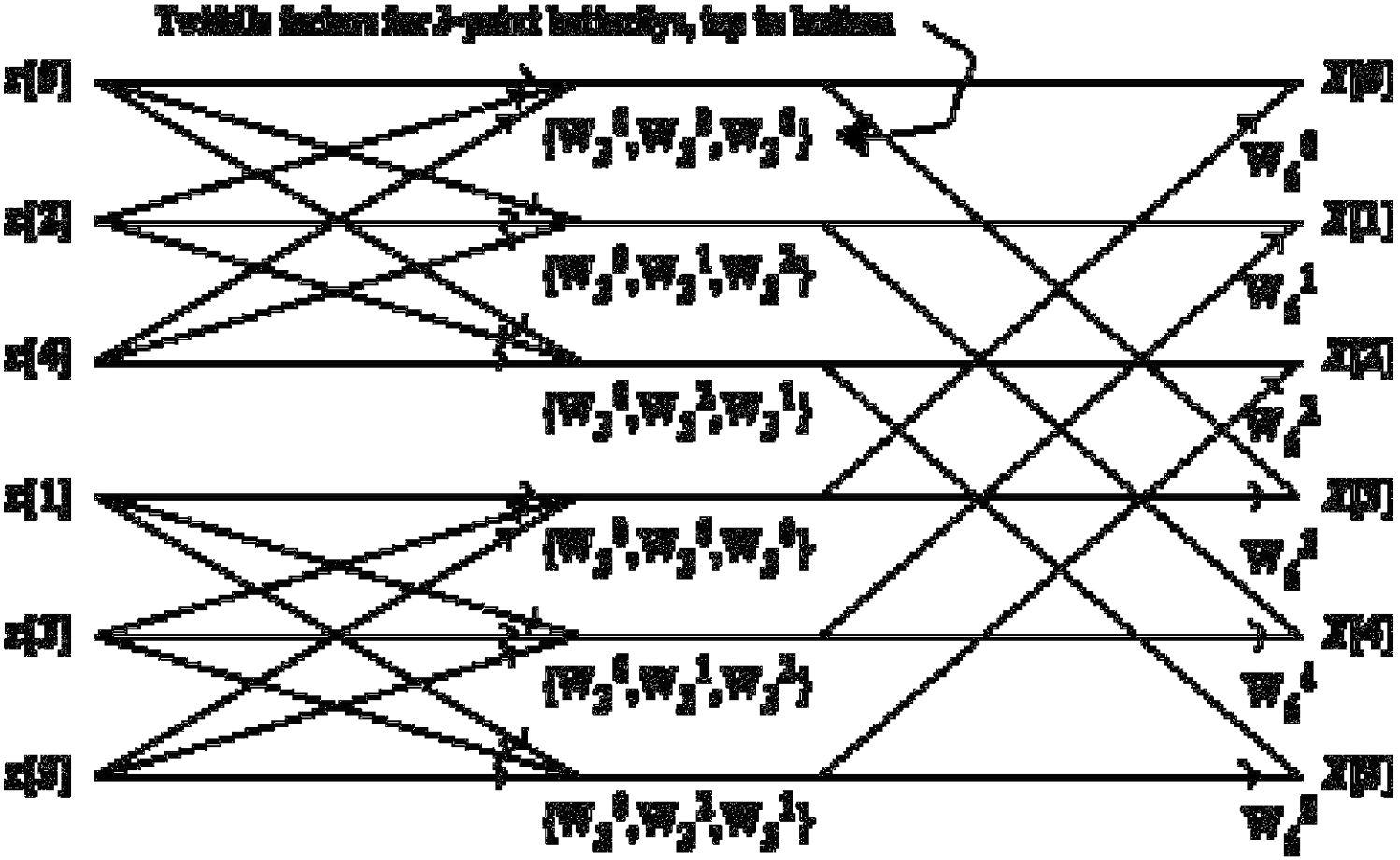
- In general, for the first decomposition we use

$$X[k] = \sum_{l=0}^{p_1-1} W_N^{lk} \sum_{r=0}^{q_1-1} x[p_1 r + l] W_{q_1}^{rk}$$

- Comments:
  - This procedure can be repeated for subsequent factors of *N*
  - The amount of computational savings depends on the extent to which *N* is "composite", able to be factored into small integers
  - Generally the smallest factors possible used, with the exception of some use of Radix-4 and Radix-8 FFTs

An Example …. The 6-point DIT FFT

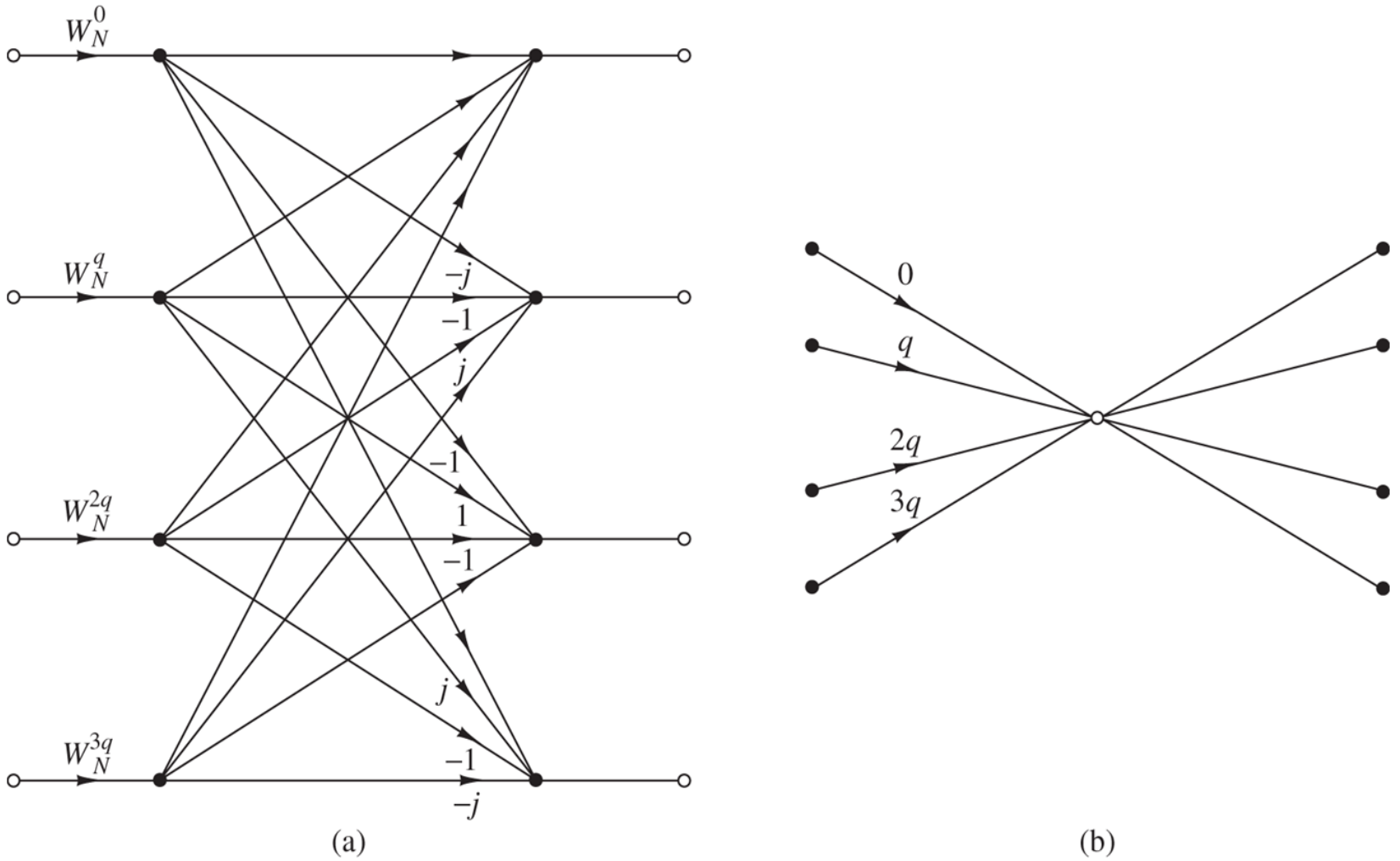
- $P_1 = 2; P_2 = 3;$ $\quad X[k] = \sum_{l=0}^{1} W_6^{lk} \sum_{r=0}^{2} x[2r+l] W_3^{rk}$



Radix-4 FFT

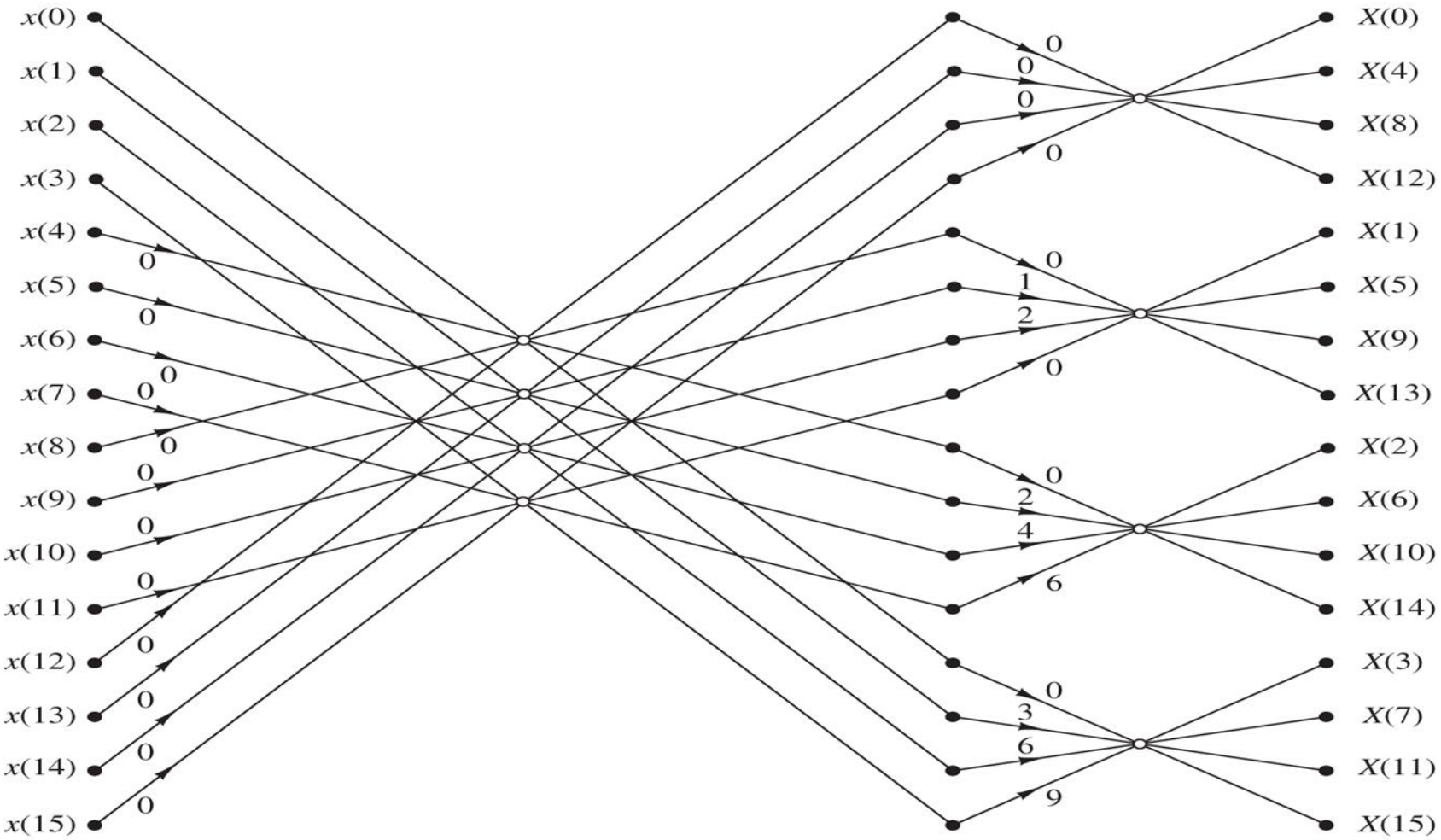
- In radix-2 you have $\log_2 N$ stages
  - Can also implement radix-4 and now have $\log_4 N$ stages
- Radix-4 Decimation-in-time: split x(n) into four time sequences instead of two
- Split x(n) into four decimated sample streams
  - $f_1(n) = x(4n)$
  - $f_2(n) = x(4n+1)$
  - $f_3(n) = x(4n+2)$
  - $f_4(n) = x(4n+3)$
  - n=0, 1, .. N/4-1
- Radix-4 Decimation-in-time (DIT) algorithm
  - In radix-4, the "butterfly" element takes in 4 inputs and produces 4 outputs
  - Butterfly implements 4-point FFT
- Computations:
  - $(3N/4)\log_4 N = (3N/8)\log_2 N$ complex multiplications → decrease from radix-2 algorithms
  - $(3N/2)\log_2 N$ complex additions → increase from radix-2 algorithms
- Downside: can only deal with FFTs of a factor of 4, such as N=4, 16, 64, 256, 1024, etc.

Radix-4 Basic Butterfly



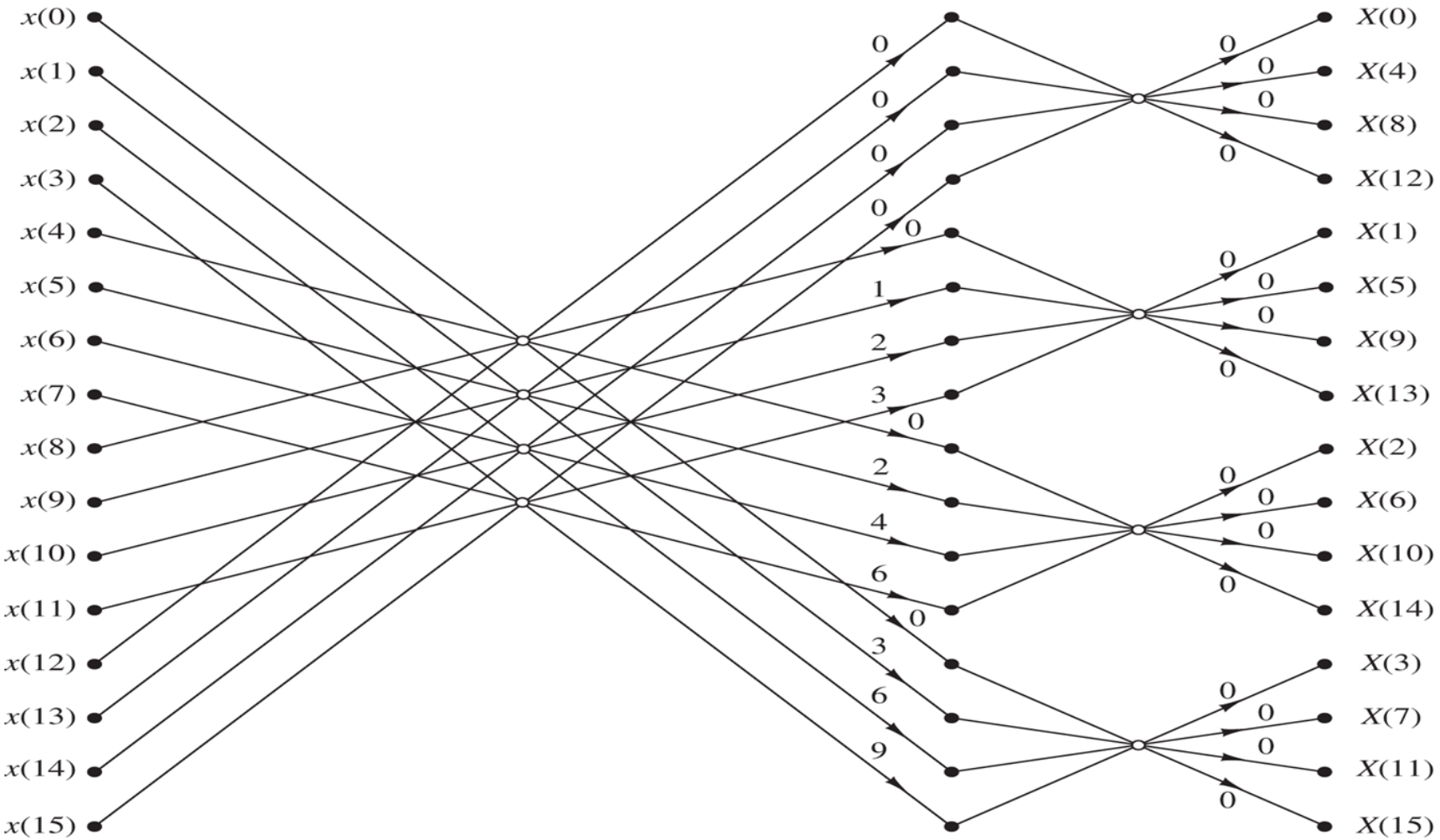
Basic butterfly computation in a radix-4 FFT algorithm.

Radix-4 Decimation-in-Time FFT (N=16)



Sixteen-point radix-4 decimation-in-time algorithm with input in normal order and output in digit-reversed order. The integer multipliers shown on the graph represent the exponents on W_{16}.

Radix-4 Decimation-in-Frequency FFT (N=16)

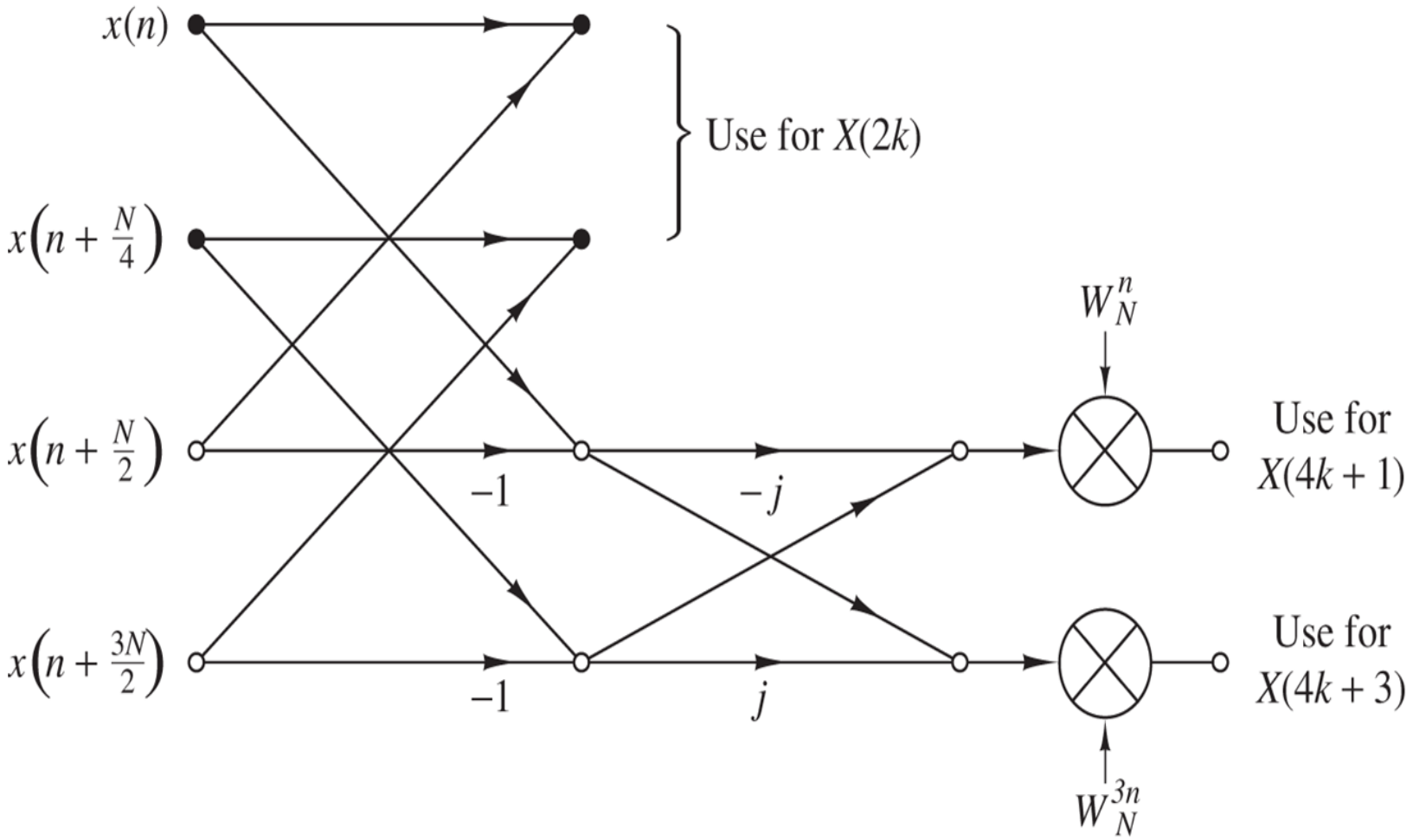


Sixteen-point, radix-4 decimation-in-frequency algorithm with input in normal order and output in digit-reversed order.

Higher Radix FFTs

- Typically do not go to radix-8 or radix-16.
  - Why?
- There do exist algorithms called split-radix algorithms which break FFT into radix-2 and radix-4 FFTs at the same time
  - Fewer number of multiplications than radix-2 or radix-4
  - However, the butterfly is irregular and so can be difficult to implement in VLSI
- There are many advanced methods of performing FFTs

Split Radix (SR) FFT Butterfly



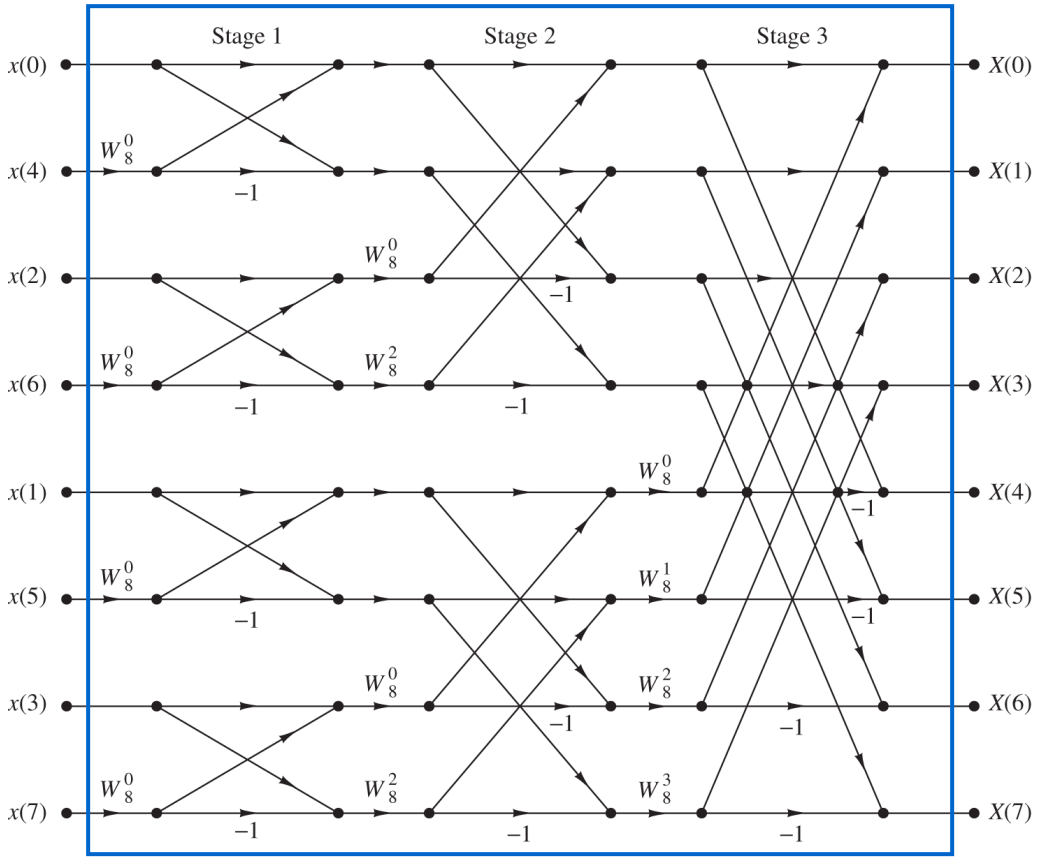
Butterfly for SRFFT algorithm.

Fast Fourier Transform Hardware Implementations

FFT Hardware Implementations

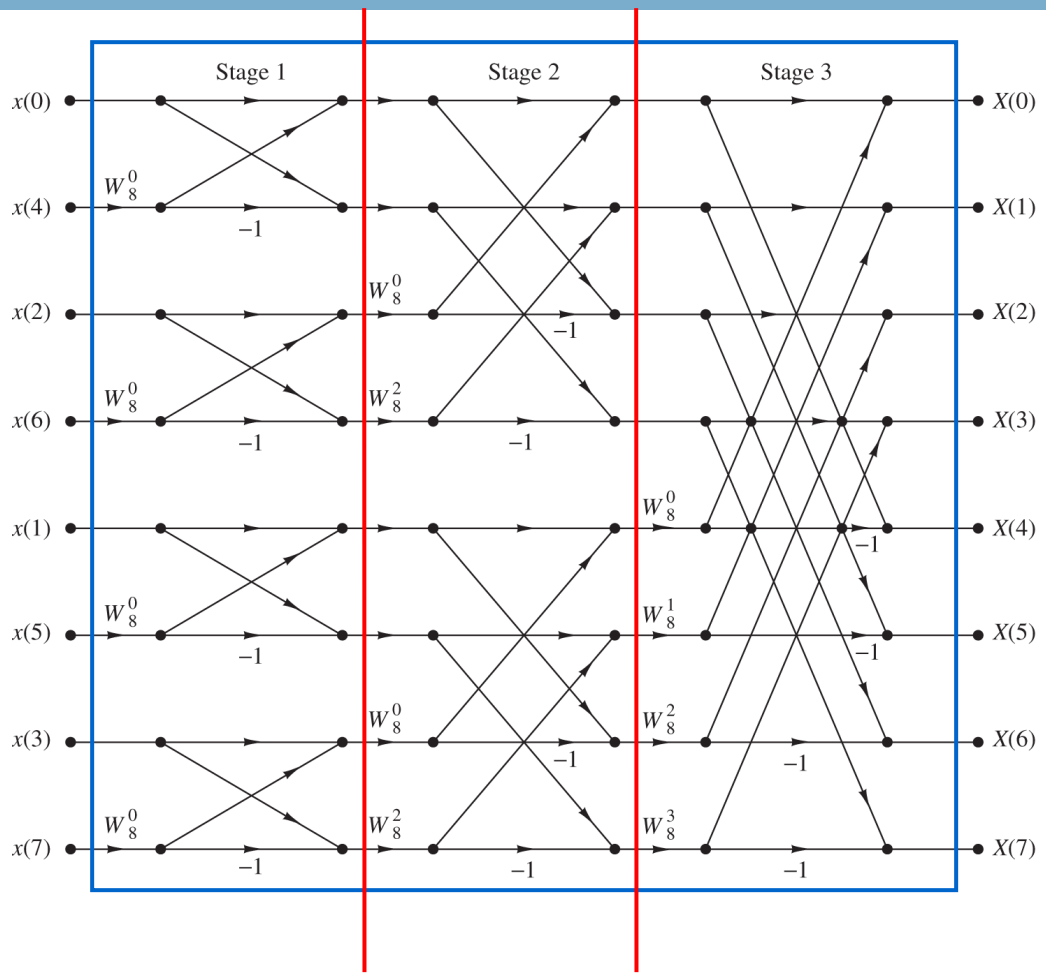
- Implementations of FFTs in hardware (FPGAs and ASICs) can generally be categorized in four categories:
  - Parallel
  - Serial/In-Place
  - Semi-Parallel
  - Pipeline
- Other implementations can be roughly placed in these categories or are a hybrid of these categories

Parallel Implementation



- Implement entire FFT structure in a parallel fashion
- Advantages: Control is easy (i.e. no controller), low latency (i.e. 0 cycles in this example), customize each twiddle factor as a multiplication by a constant
- Disadvantages: Huge Area, Routing congestion

Parallel Implementation with Pipelining

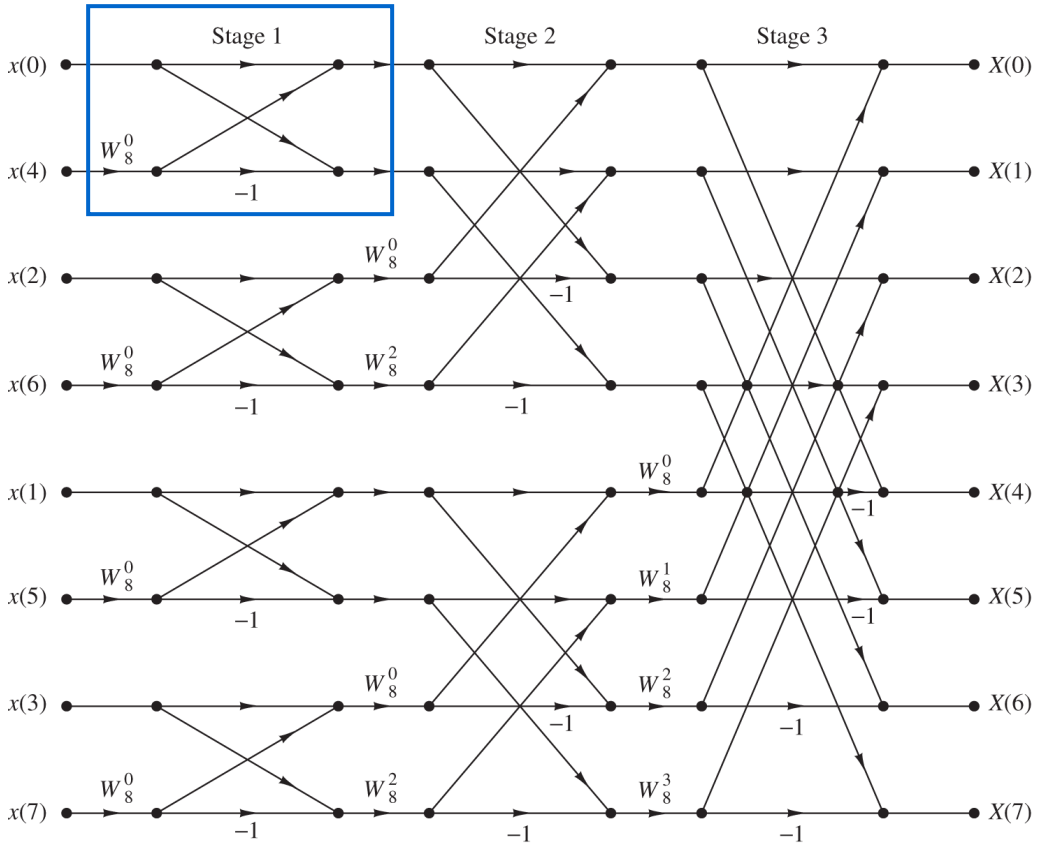


Add cutset pipeline stages here.

Due to feedforward structure, can pipeline as much as desired.

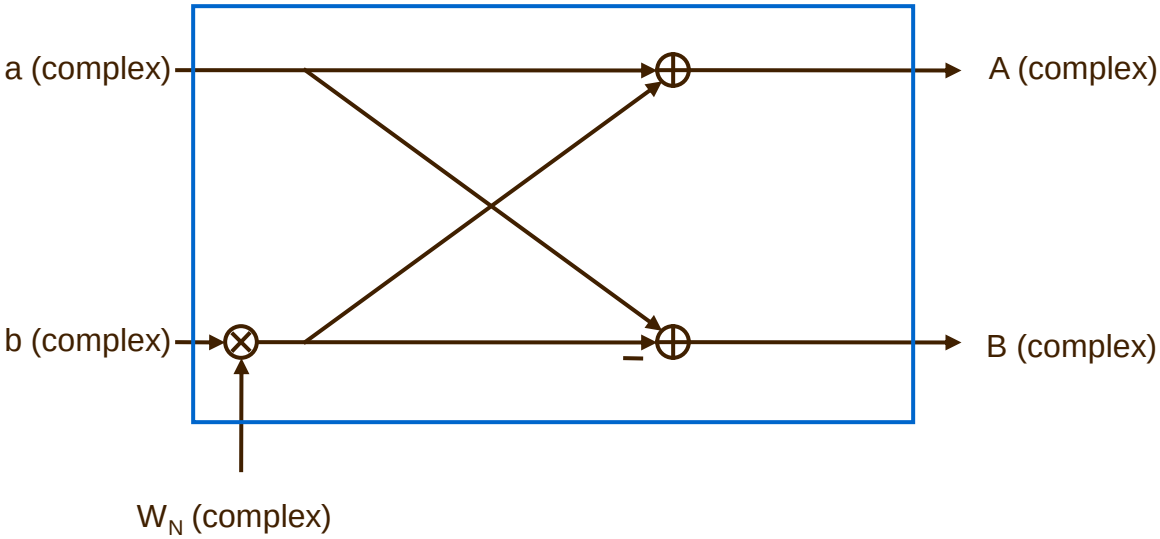
- This example breaks critical path into a single butterfly (i.e. complex multiply and a complex add)

Serial/In-Place FFT Implementation

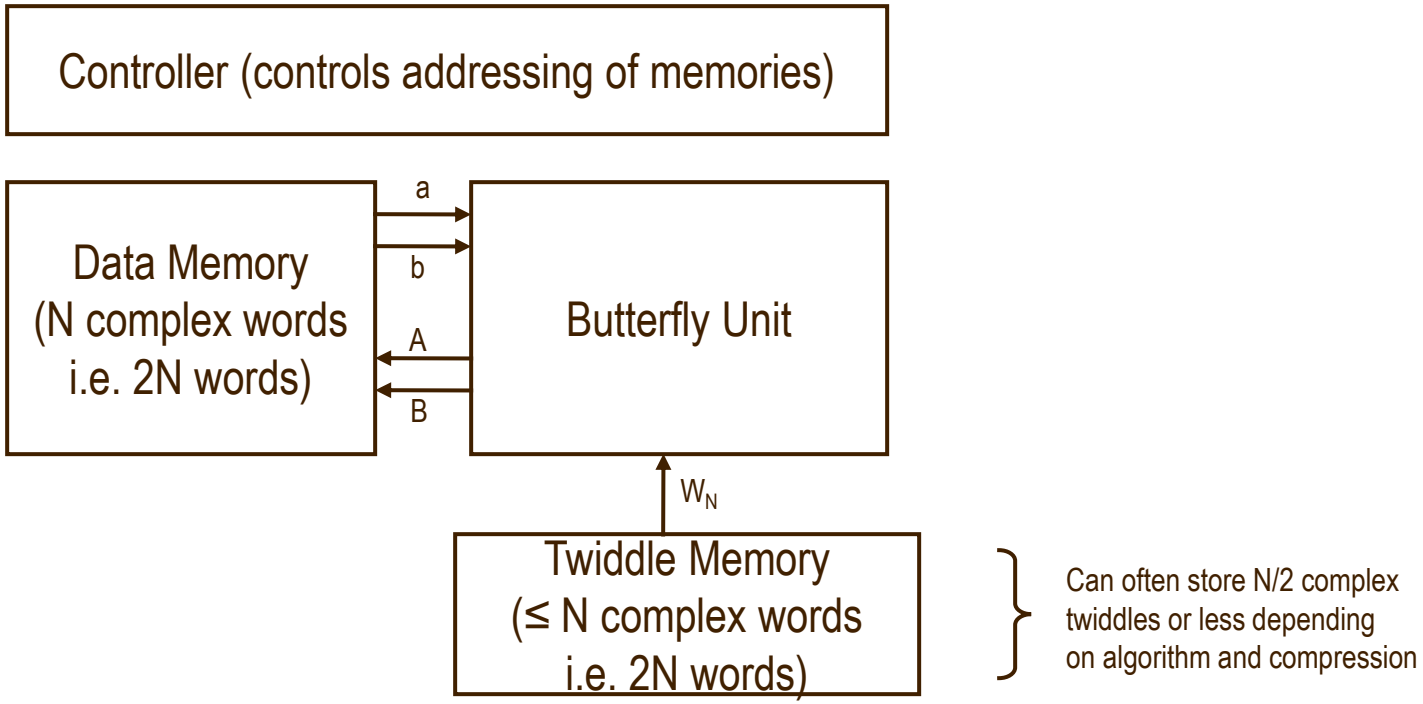


- Implement a single butterfly. Use that butterfly and some memory to compute entire FFT
- Advantages: Small Area
- Disadvantages: Large Latency, Complex Controller

Serial/In-Place FFT Butterfly

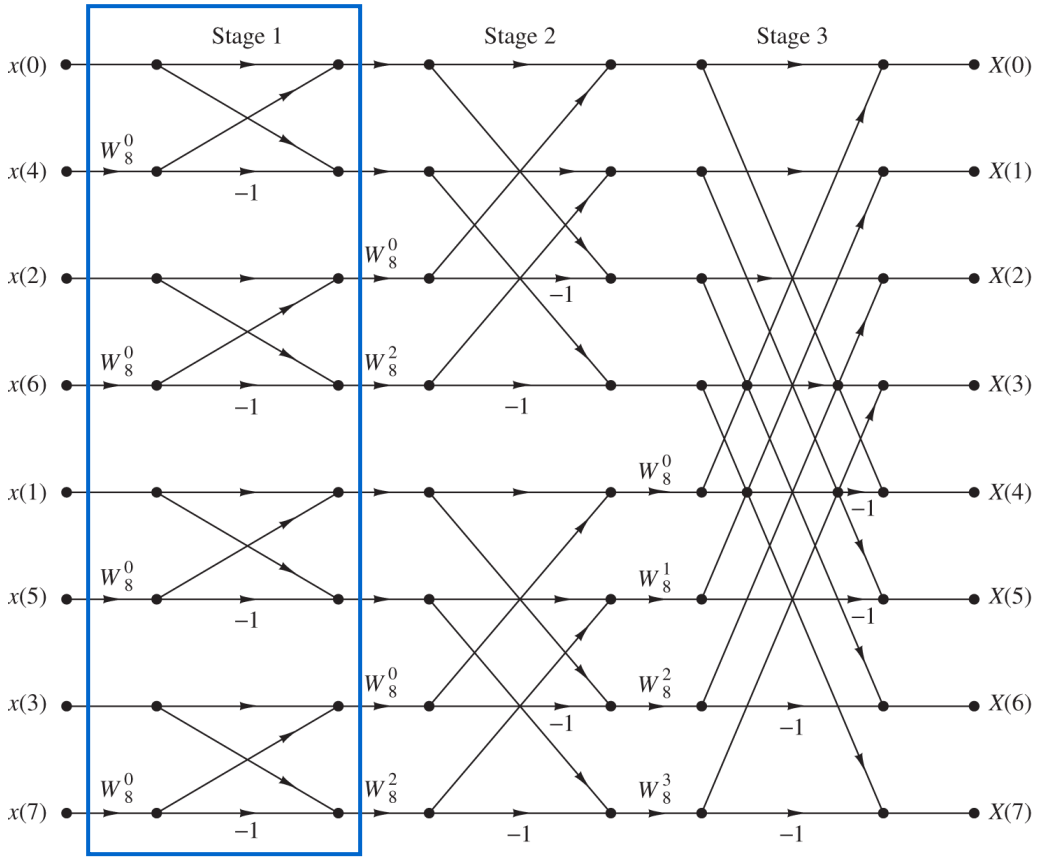


Serial/In-Place FFT



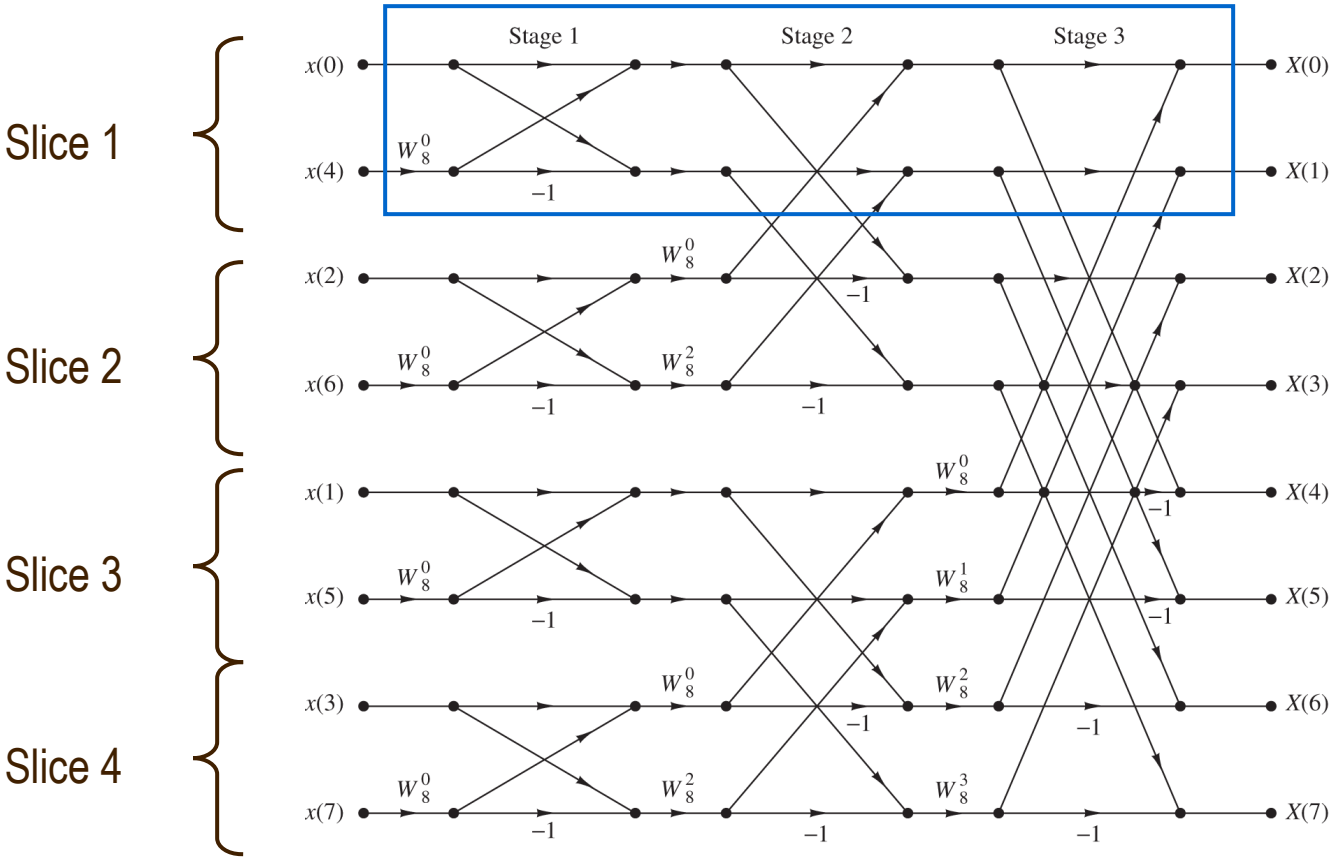
- Operation:
  - Put input data into data memory
  - Compute butterfly operation, after which OVERWRITE previous data words (since they are no longer needed)
  - Move to next butterfly
- Example for N=8
  - Do x(0) and x(4) butterfly, store results in x(0) and x(4) registers
  - Do x(2) and x(6) butterfly, store results in x(2) and x(6) registers
  - To complete stage 1, requires (N/2) butterfly operations
  - Do this for each stage (i.e. stage 1, stage 2, stage 3 for N=8), total of log2N butterfly operations per stage
- Total latency: (N/2) log2N cycles to complete an N-point FFT (assuming one butterfly per cycle)
- There are many ways to "compress" the twiddle factor memory to take advantage of symmetries and subexpression sharing

Semi-Parallel FFT Implementation



- Implement an entire stage and use memory to store intermediate results (using same override technique as in serial/in-place FFTs)
- Advantages: Less area than full parallel
- Disadvantages: More complex controller than full parallel, no advantage of fixed multipliers, longer latency than full parallel

Pipeline FFT



- Pipeline FFT is very common for communication systems (OFDM, DMT)
- Implements an entire "slice" of the FFT and reuses-hardware to perform other slices
- Advantages: Particularly good for systems in which x(n) comes in serially (i.e. no block assembly required), very fast, more area efficient than parallel, can be pipelined
- Disadvantages: Controller can become complicated, large intermediate memories may be required between stages, latency of N cycles (more if pipelining introduced)

Pipeline FFT Classification

- Pipeline FFTs can be classified in two groups:
  - Feed-Forward Structures (Commutator)
  - Feedback Structures
- Can also be categorized according to radix:
  - Radix-2 Structures
  - Radix-4 Structures
  - etc.
- The key to pipeline FFTs is the utilization factor (i.e. want all computations elements active as much as possible)
  - High utilization factor means higher pipeline FFT efficiency