Case Study: FFT

Shao-Yi Chien

Outline

- Introduction to FFT
- Algorithm of FFT
- Architectures of FFT

Outline

- Introduction to FFT
- Algorithm of FFT
- Architectures of FFT

Introduction to FFT

- **FFT: Fast Fourier Transform**
- Fast algorithm for DFT (digital Fourier transform)
- The most famous one is proposed by Cooley and Turkey in 1965
- The complexity of DFT: N^2
- The complexity of FFT: NlogN

Outline

- Introduction to FFT
- Algorithm of FFT
- Architectures of FFT

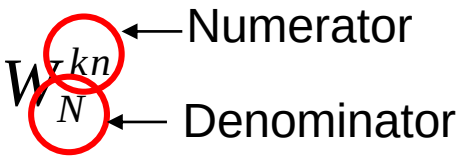
Algorithm of FFT

- Start from DFT and the twiddle factors
- Decimation-in-time FFT algorithms
- Decimation-in-frequency FFT algorithms
- Radix-4 FFT algorithm
- Radix-n FFT algorithm

Start from DFT and the Twiddle Factors

- N-point DFT
  - $$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j(2\pi nk/N)} \quad k = 0,1,\ldots,N-1$$
  - The complexity is N^2
- Define the **twiddle factor**

$$W_N^{kn} = e^{-j(2\pi nk/N)}$$



- Properties of the twiddle factor
  - Complex conjugate symmetry
  - Periodicity in n and k

$$W_N^{k[N-n]} = W_N^{-kn} = (W_N^{kn})^*$$

$$W_N^{kn} = W_N^{k(n+N)} = W_N^{(k+N)n}$$

Decimation-in-time FFT algorithms

- To decompose the sequence x[n] into successively smaller sequences

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

$$=\sum_{n\ \text{even}}x[n]W_N^{nk}+\sum_{n\ \text{odd}}x[n]W_N^{nk}$$

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

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

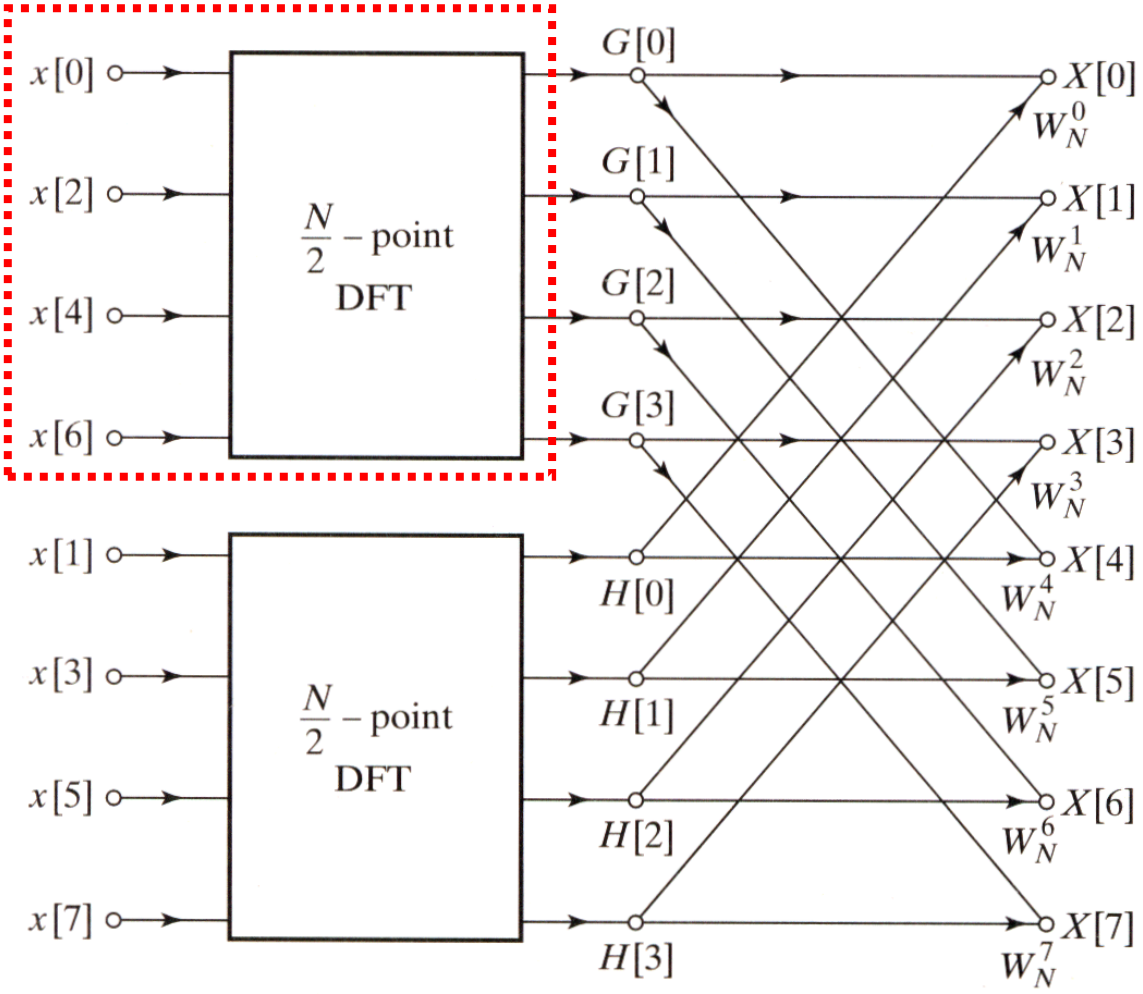
$$=\sum_{r=0}^{(N/2)-1}x[2r]W_{N/2}^{rk}+W_N^k\sum_{r=0}^{(N/2)-1}x[2r+1]W_{N/2}^{rk}=G[k]+W_N^kH[k]$$

Note

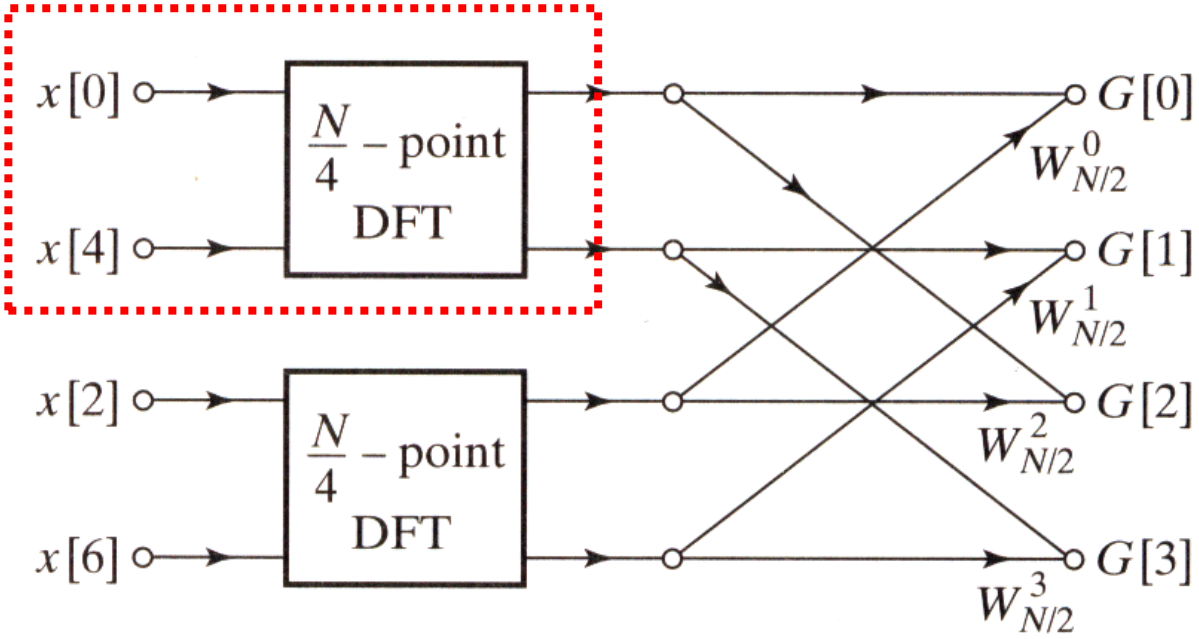
$G[k+N/2]=G[k]$

$H[k+N/2]=H[k]$

Flow Graph if N=8



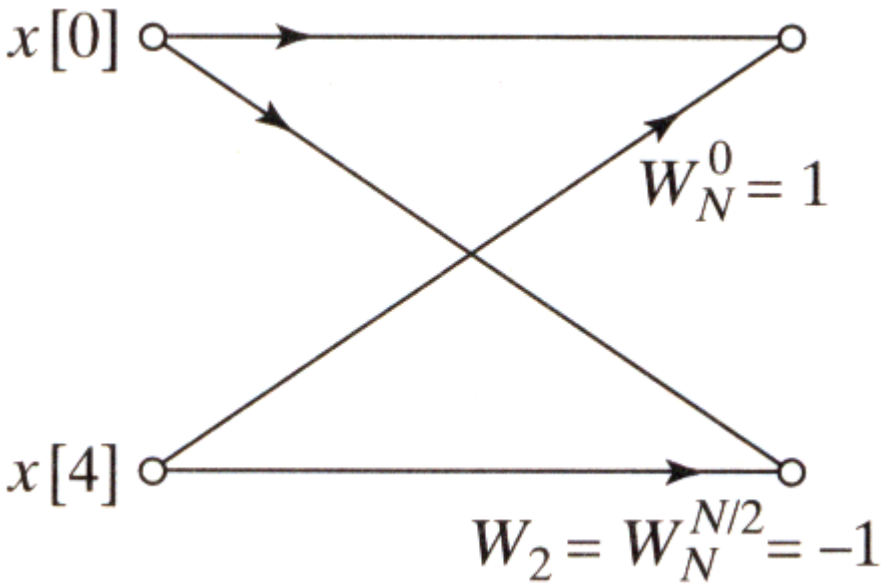
Flow Graph if N=4



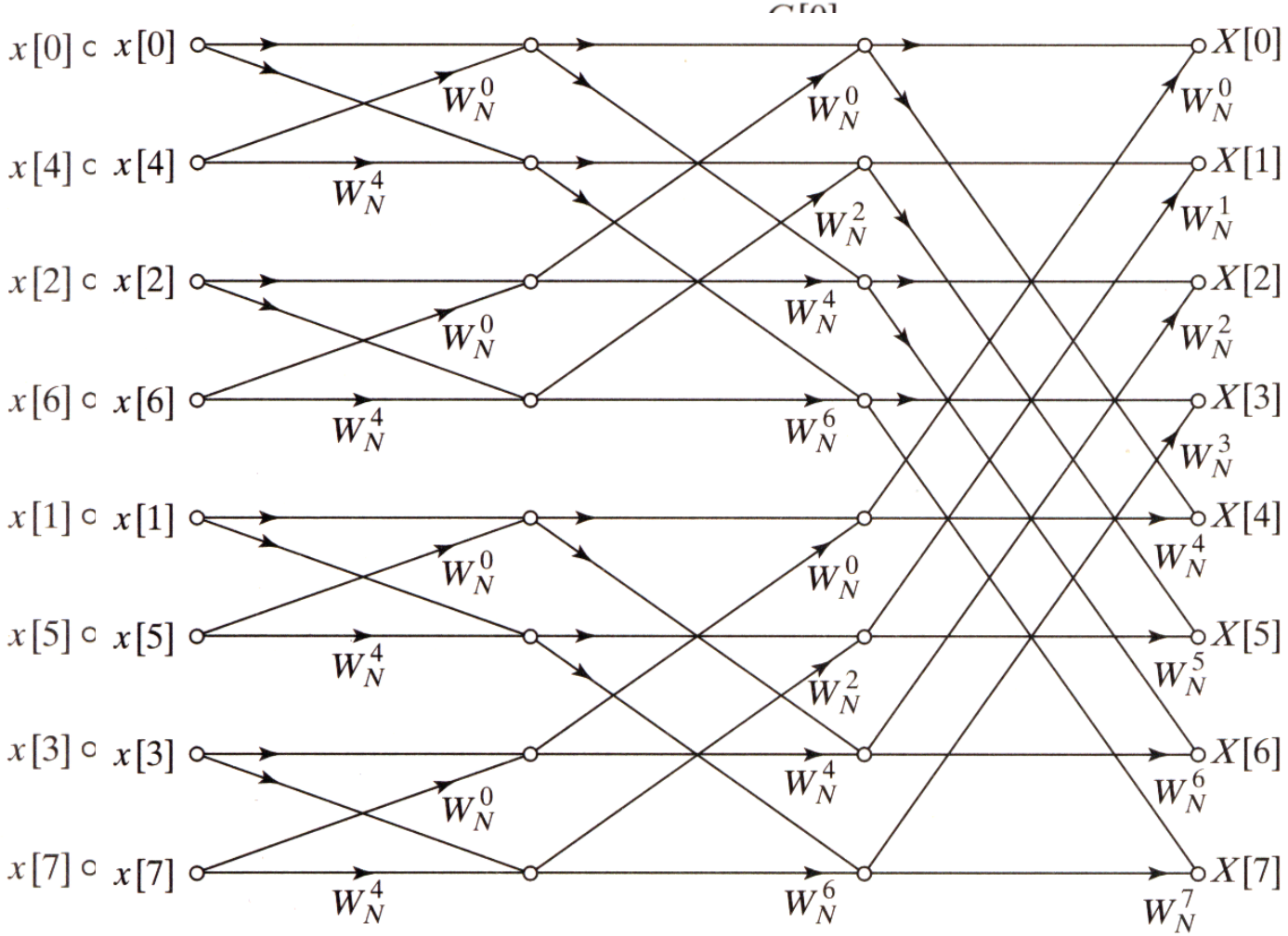
$$G[k] = \sum_{l=0}^{(N/4)-1} g[2l] W_{N/4}^{lk} + W_{N/2}^{k} \sum_{l=0}^{(N/4)-1} g[2l+1] W_{N/4}^{lk}$$

$$H[k] = \sum_{l=0}^{(N/4)-1} h[2l] W_{N/4}^{lk} + W_{N/2}^{k} \sum_{l=0}^{(N/4)-1} h[2l+1] W_{N/4}^{lk}$$

Flow Graph if N=2

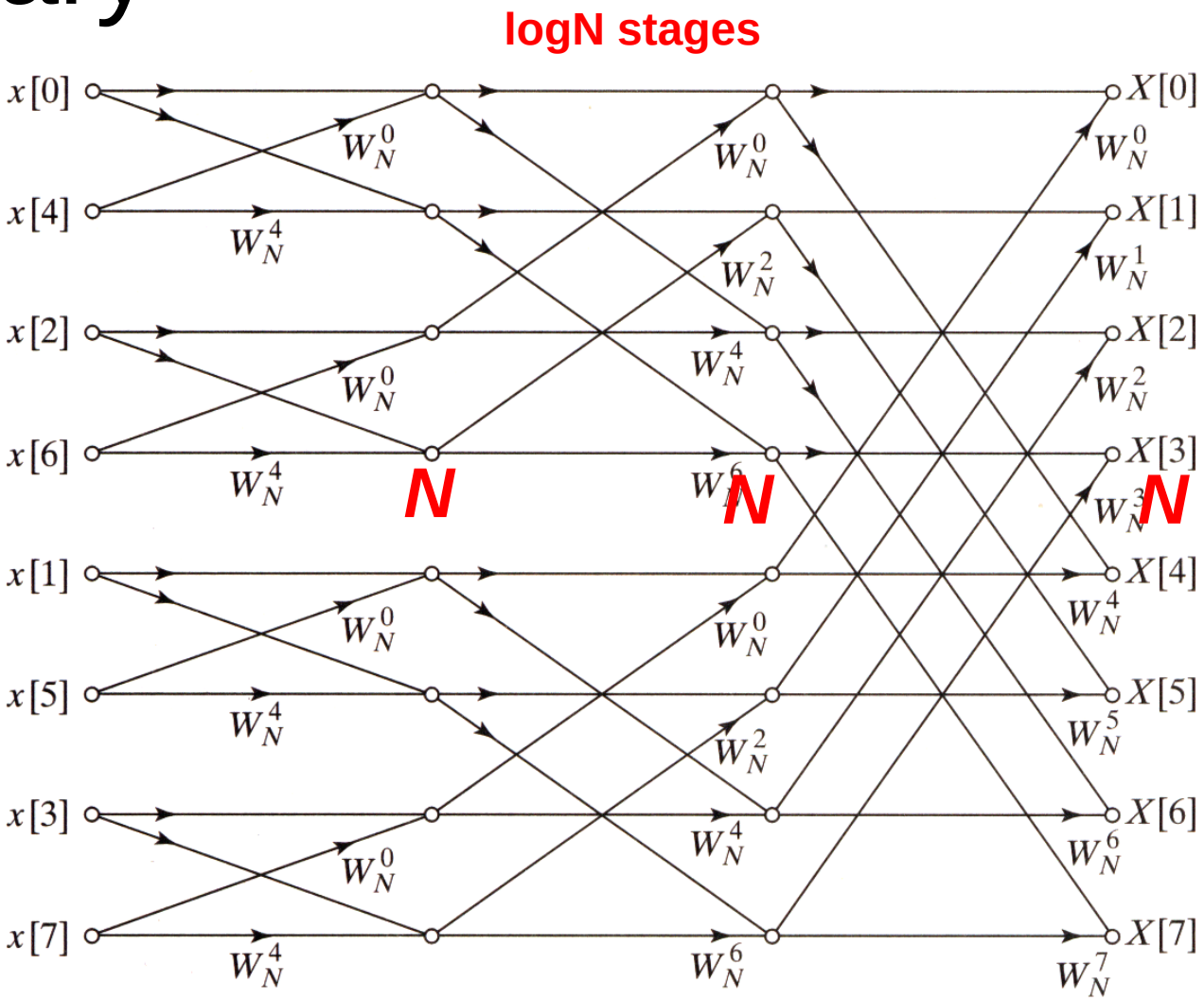


Summary

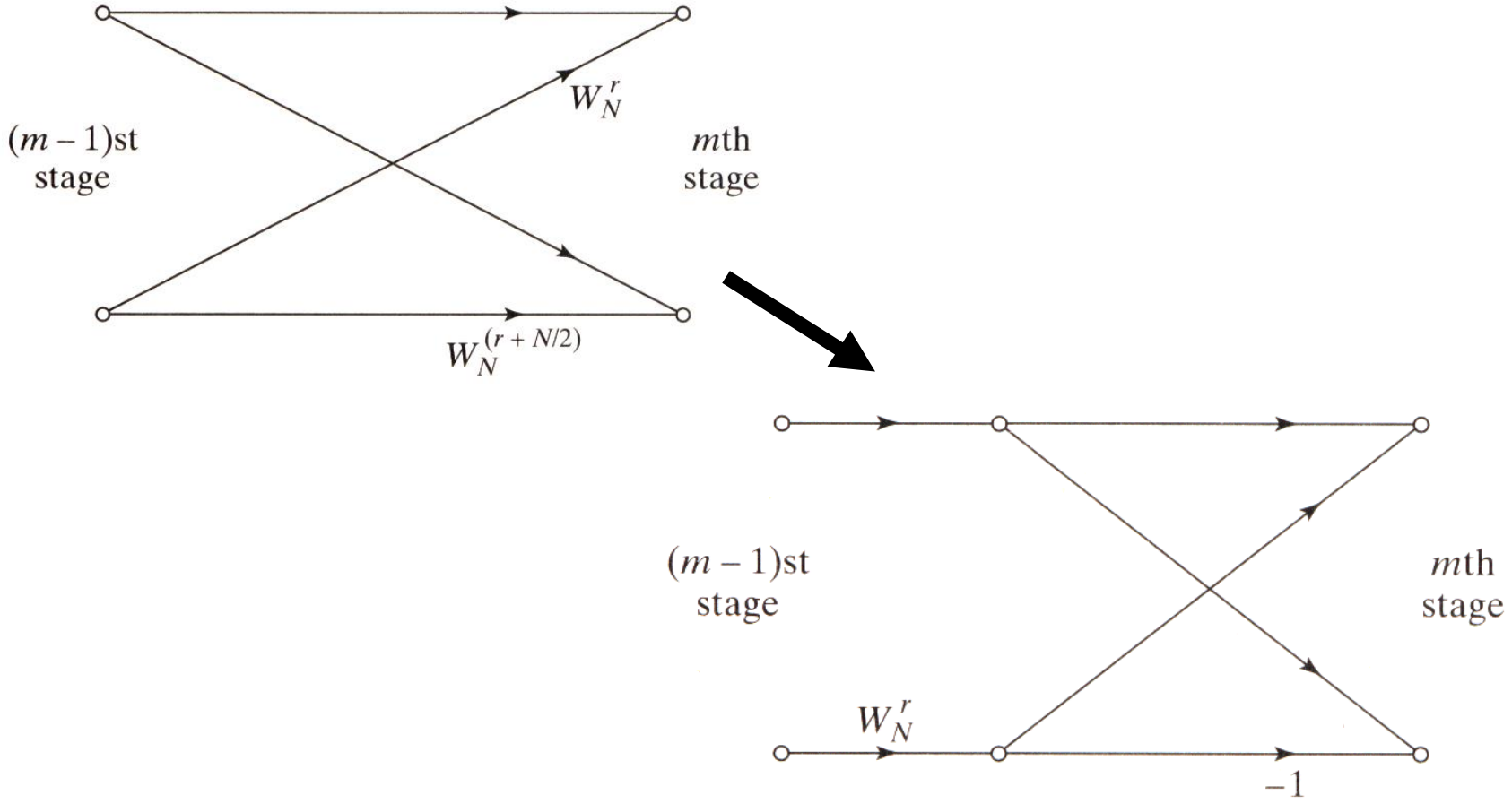

x[0] = x[0]
x[4] = x[4]
x[2] = x[2]
x[6] = x[6]
x[1] = x[1]
x[5] = x[5]
x[3] = x[3]
x[7] = x[7]
X[0]
X[1]
X[2]
X[3]
X[4]
X[5]
X[6]
X[7]
W_N^0
W_N^4
W_N^2
W_N^6
W_N^1
W_N^3
W_N^5
W_N^7

Summary

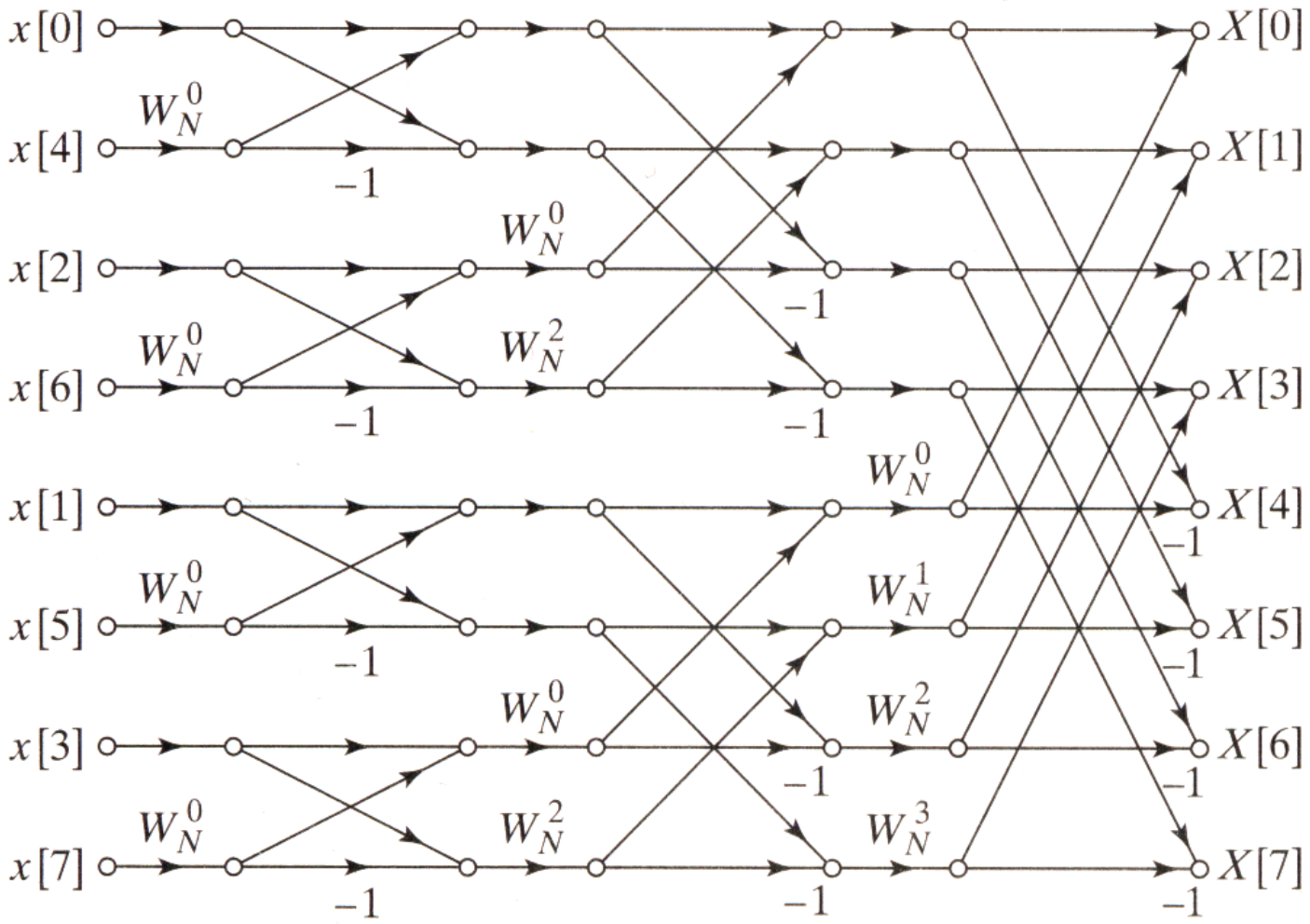
- Computation: **NlogN**



Basic Butterfly Computation

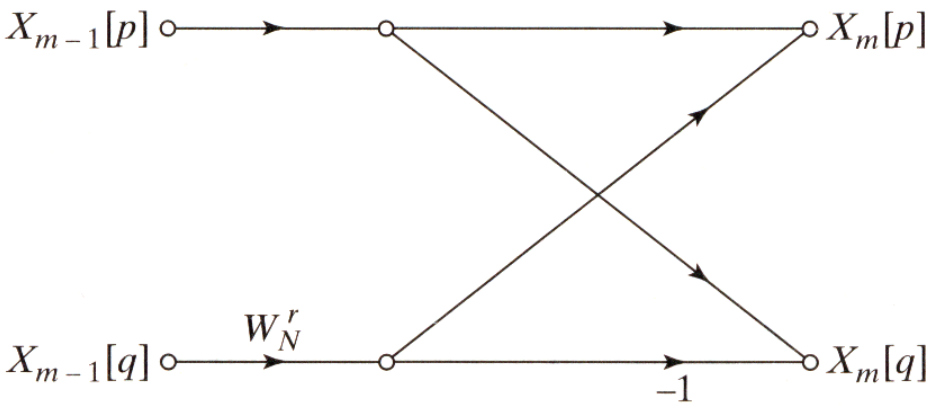


Final Flow Graph of 8-Point FFT

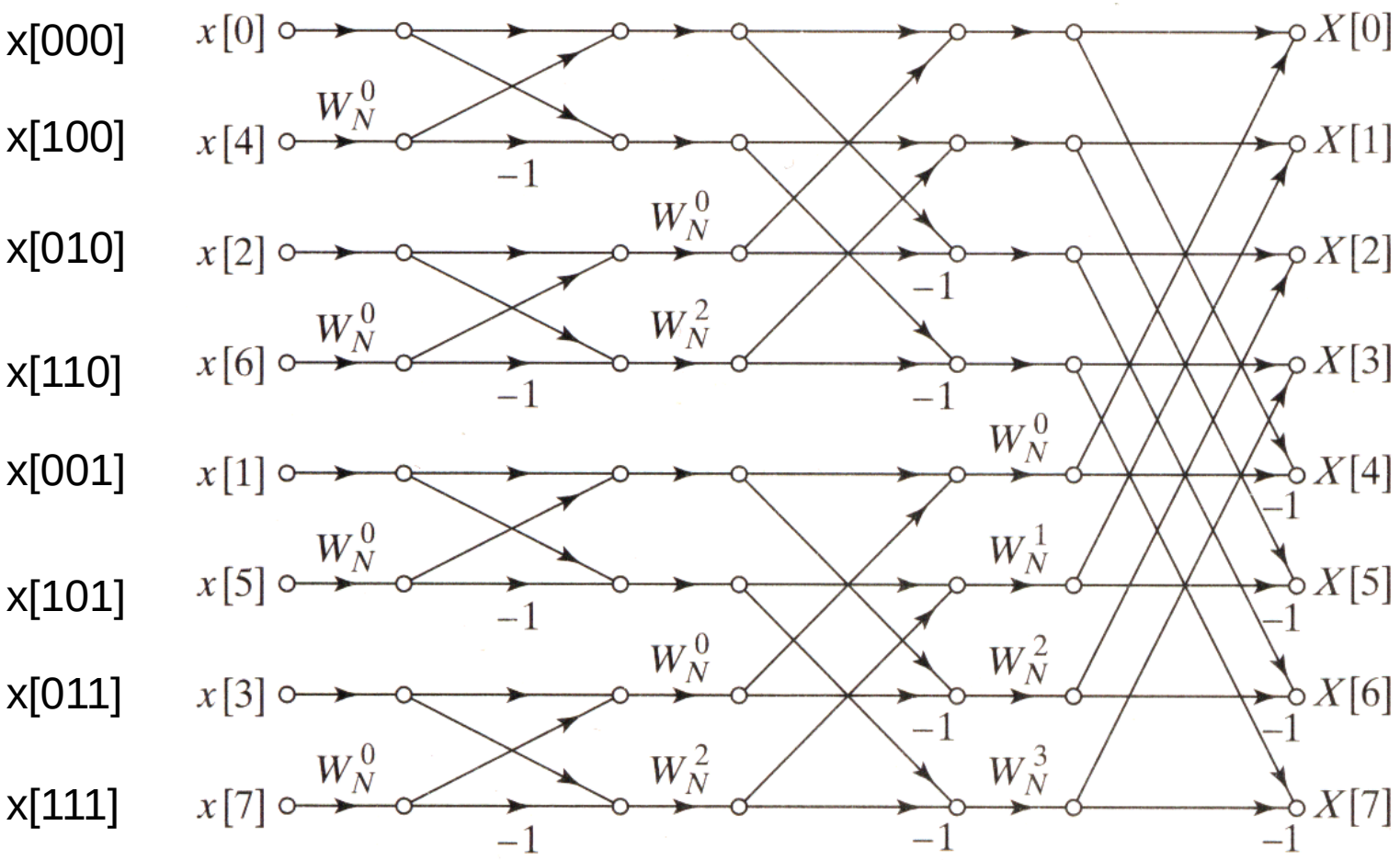


In-Place Computation

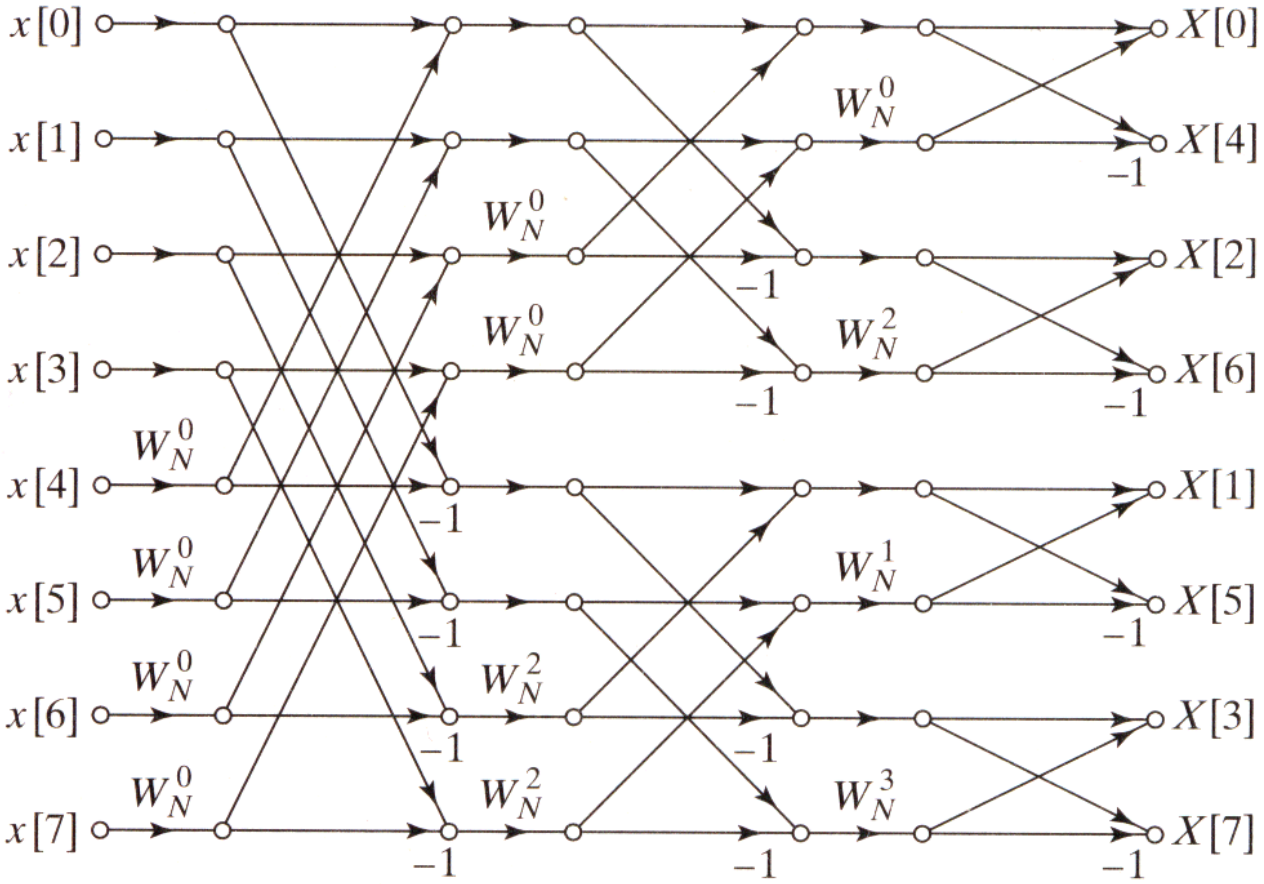
- The computation can be finished locally in a butterfly
- For one stage, we can use N registers to store the results instead of 2N registers



Bit-Reversed Order

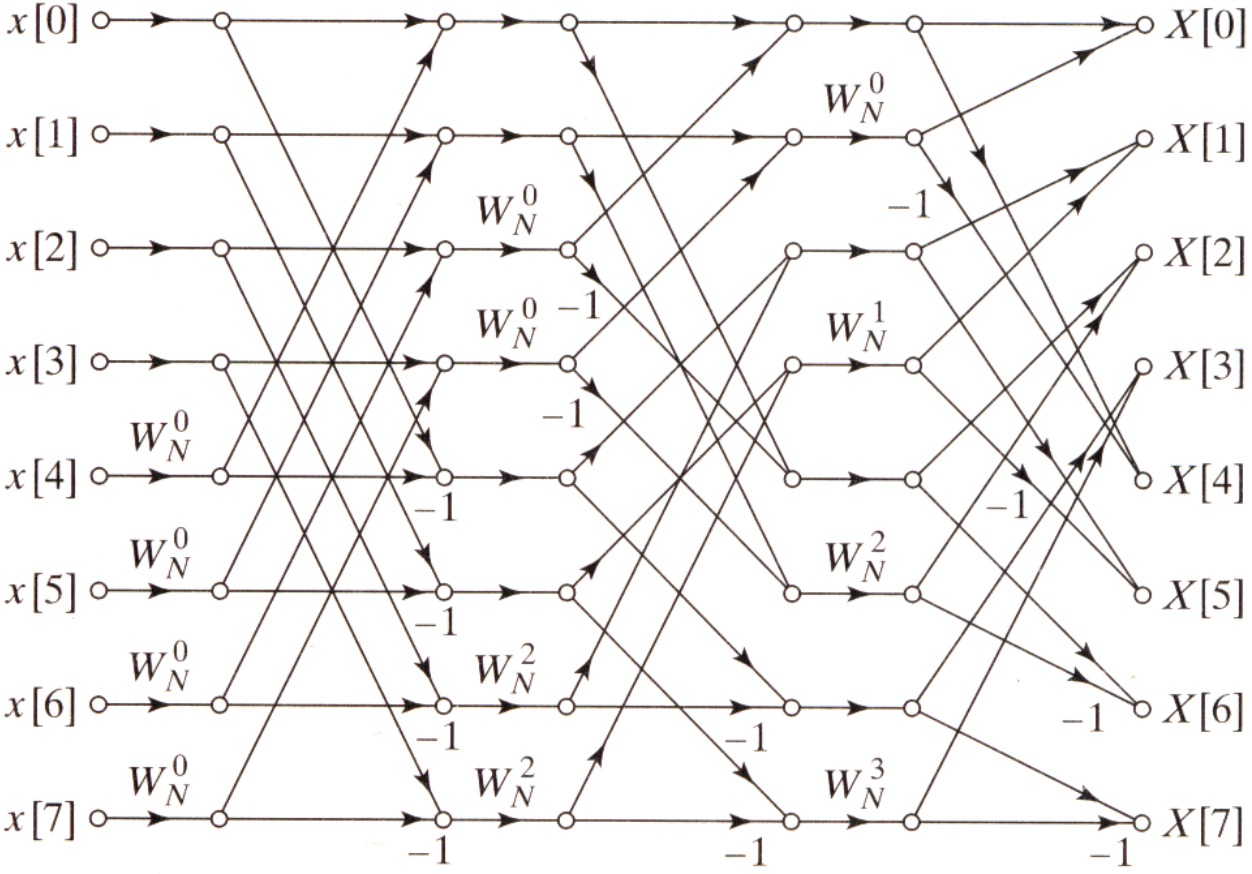


Alternative Forms (1/3)



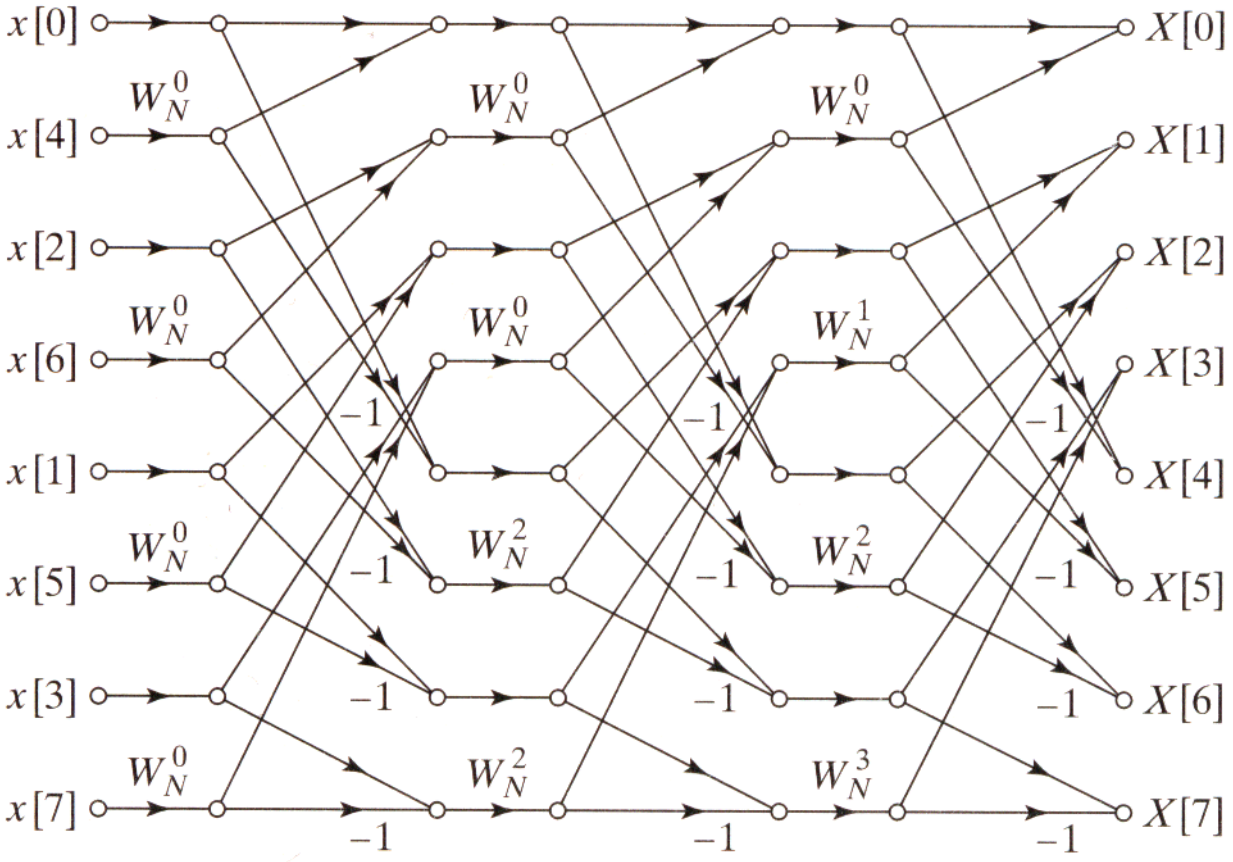
- Input in normal order, output in bit-reversed order
- For hardware architecture design, need random access memory

Alternative Forms (2/3)



- Loss the feature of in-place operation
  - Need 2N registers to buffer the partial results
- Irregular access

Alternative Forms (3/3)



- Can use sequential access memory
- Good for large transform
- The connection of every stage is the same

Decimation-in-Frequency FFT Algorithms (1/2)

- Divide the output sequence X[k] into smaller and smaller subsequences

Even: $X[2r] = \sum_{n=0}^{N-1} x[n] W_N^{n(2r)} \quad r = 0,1,\ldots,(N/2)-1$

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

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

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

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

$$W_N^{2r[n+(N/2)]} = W_N^{2rn} W_N^{rN} = W_N^{2rn}$$

Decimation-in-Frequency FFT Algorithms (2/2)

Odd: $X[2r+1] = \sum_{n=0}^{N-1} x[n] W_N^{n(2r+1)} \quad r = 0,1,\ldots,(N/2)-1$

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

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

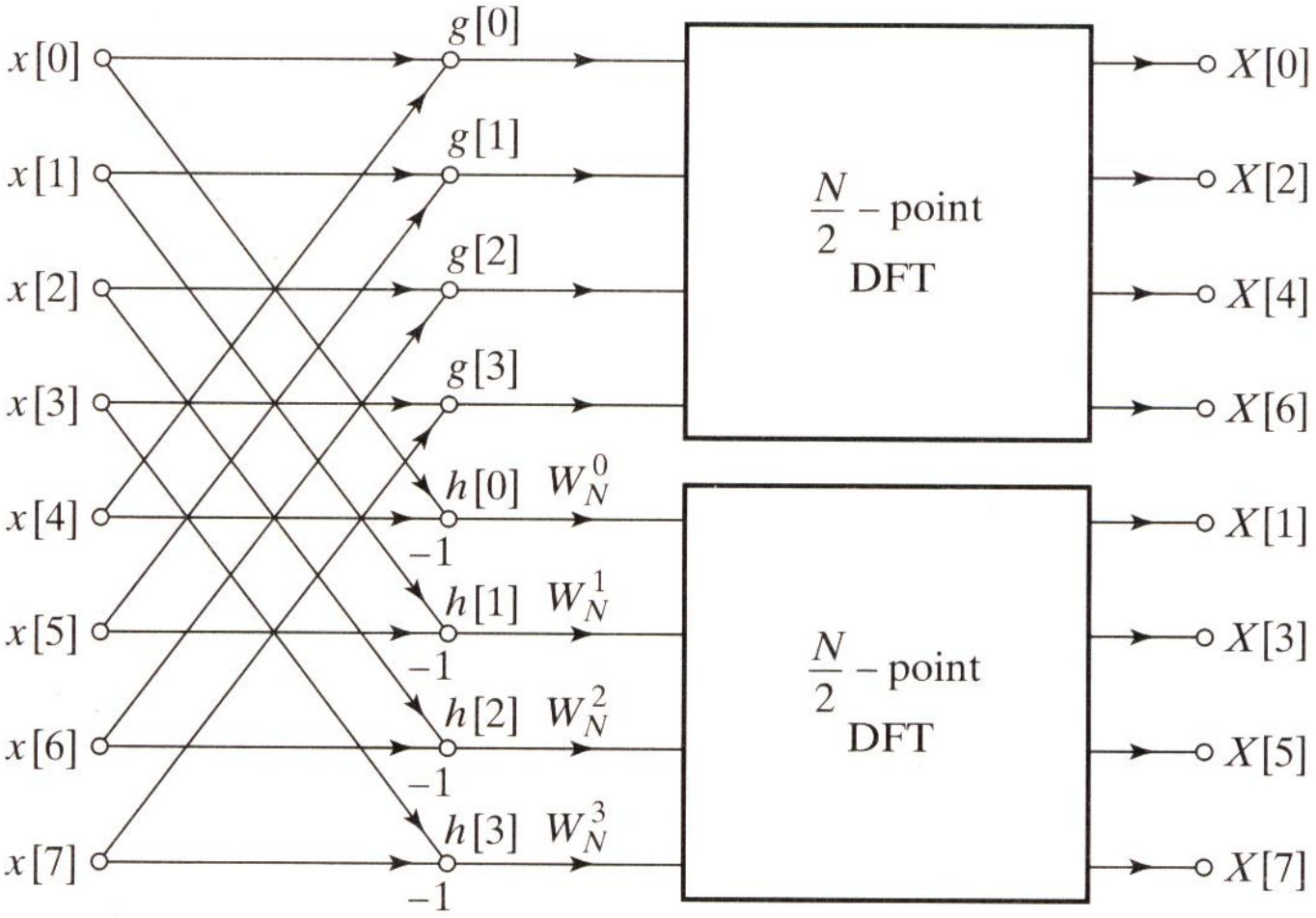
$$= \sum_{n=0}^{(N/2)-1} x[n] W_N^{n(2r+1)} - \sum_{n=0}^{(N/2)-1} x[n+(N/2)] W_N^{n(2r+1)}$$

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

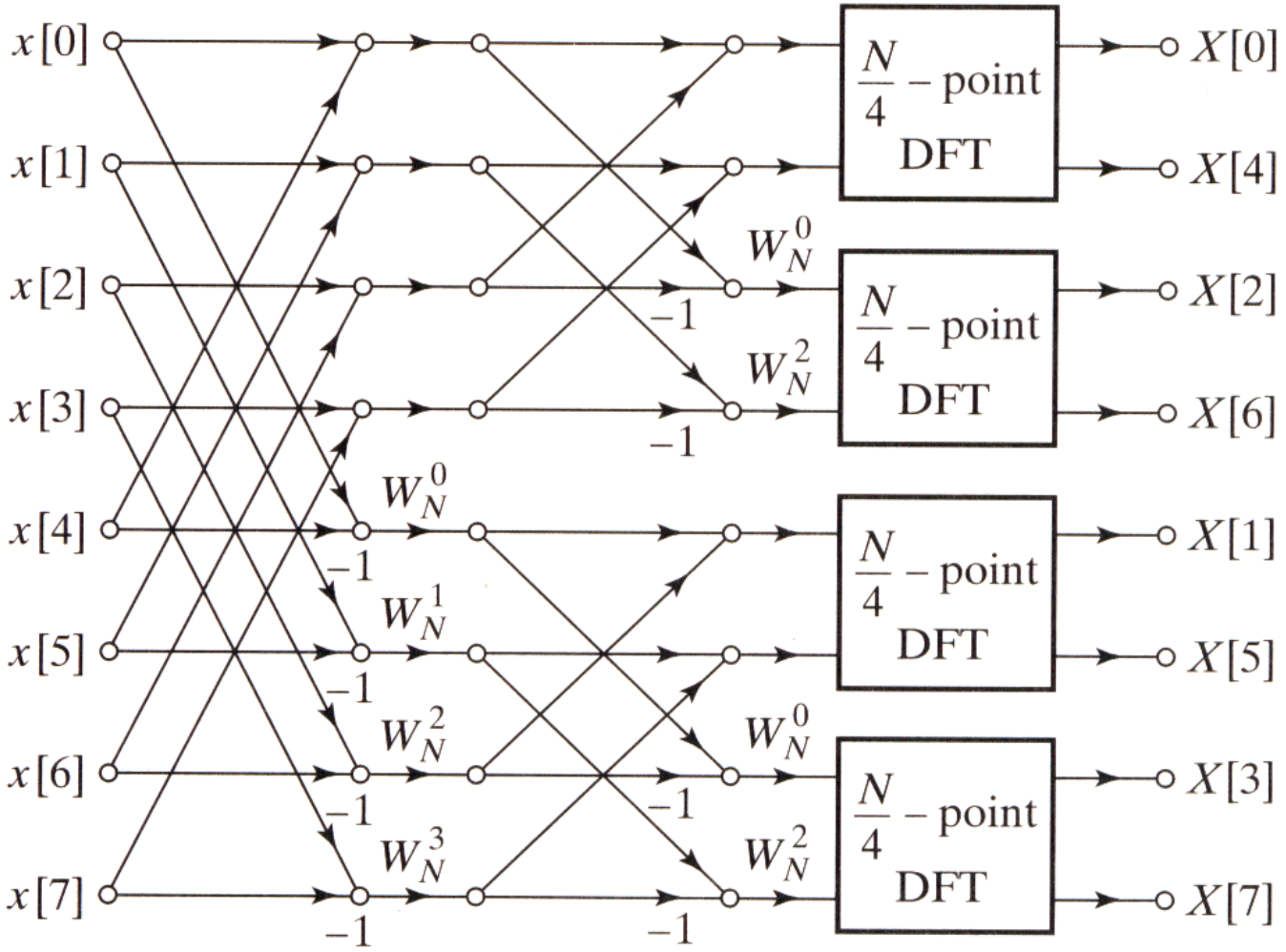
$$= \sum_{n=0}^{(N/2)-1} (x[n] - x[n+(N/2)]) W_N^{n} W_{N/2}^{nr} = \sum_{n=0}^{(N/2)-1} (h[n] W_N^{n}) W_{N/2}^{nr}$$

$$W_N^{[n+(N/2)](2r+1)} = W_N^{(N/2)(2r+1)} W_N^{n(2r+1)} = -W_N^{n(2r+1)}$$

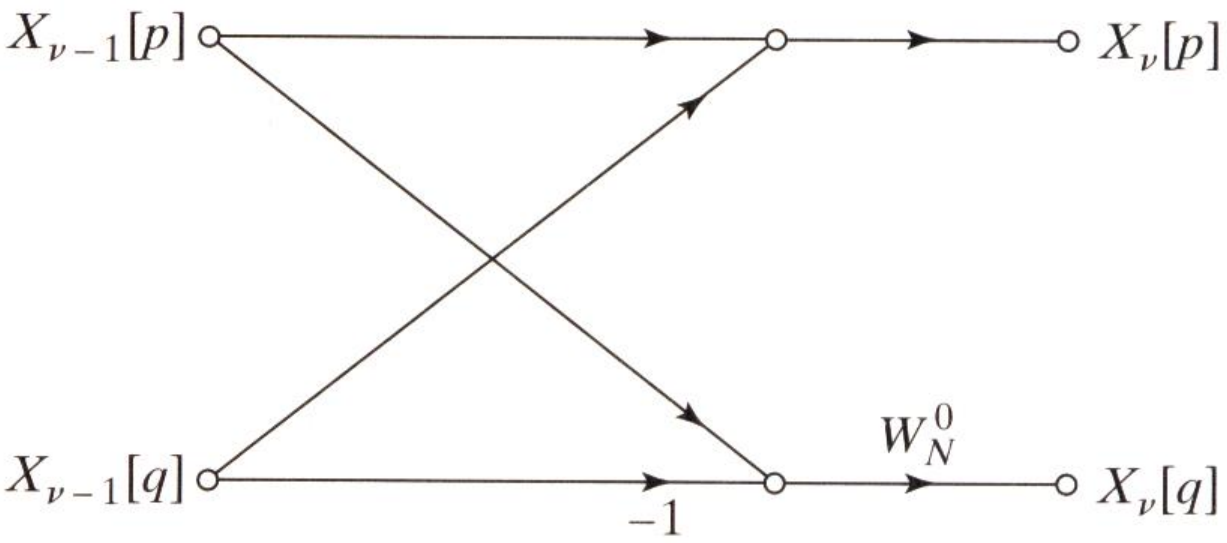
Flow Graph if N=8



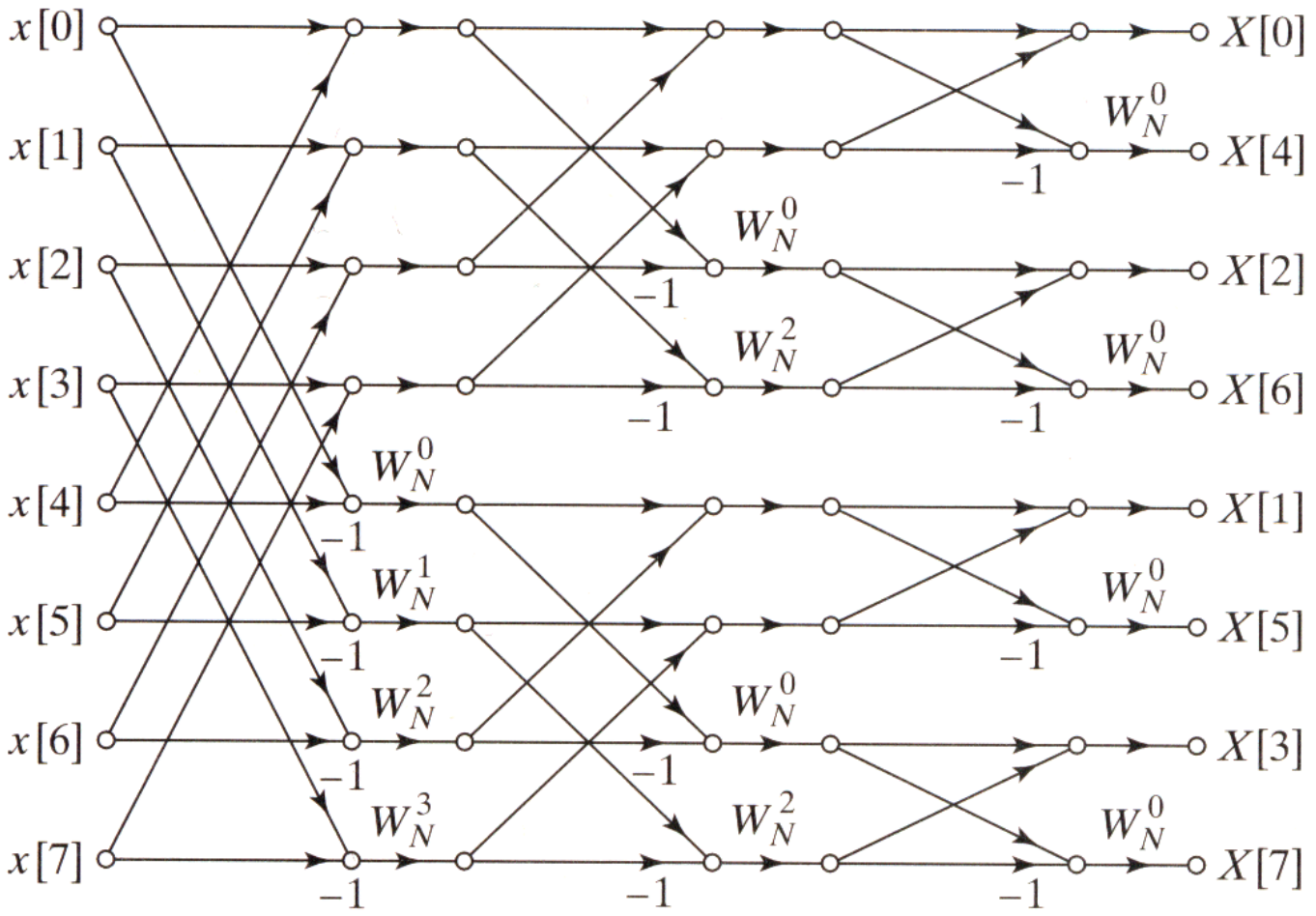
Flow Graph if N=4



Flow Graph if N=2

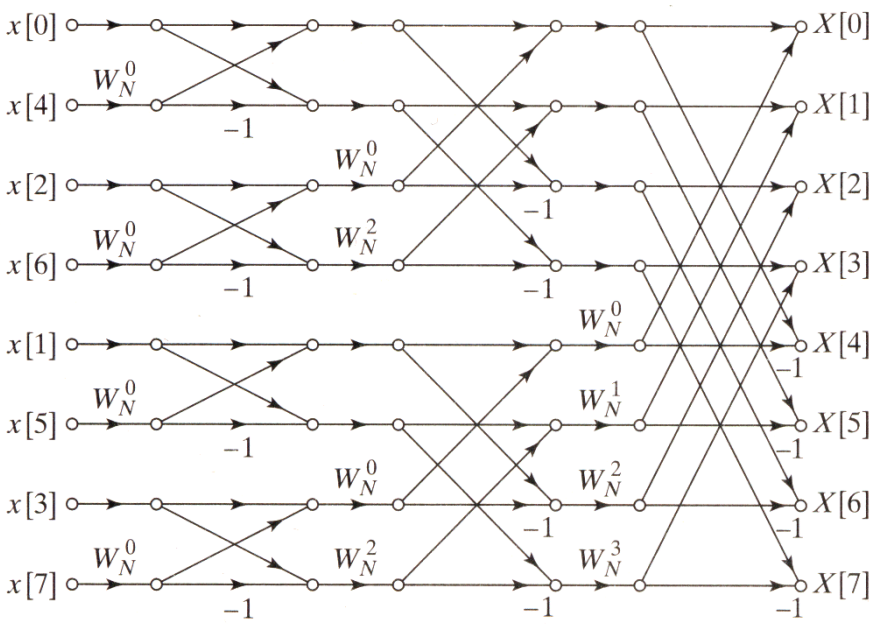


Summary

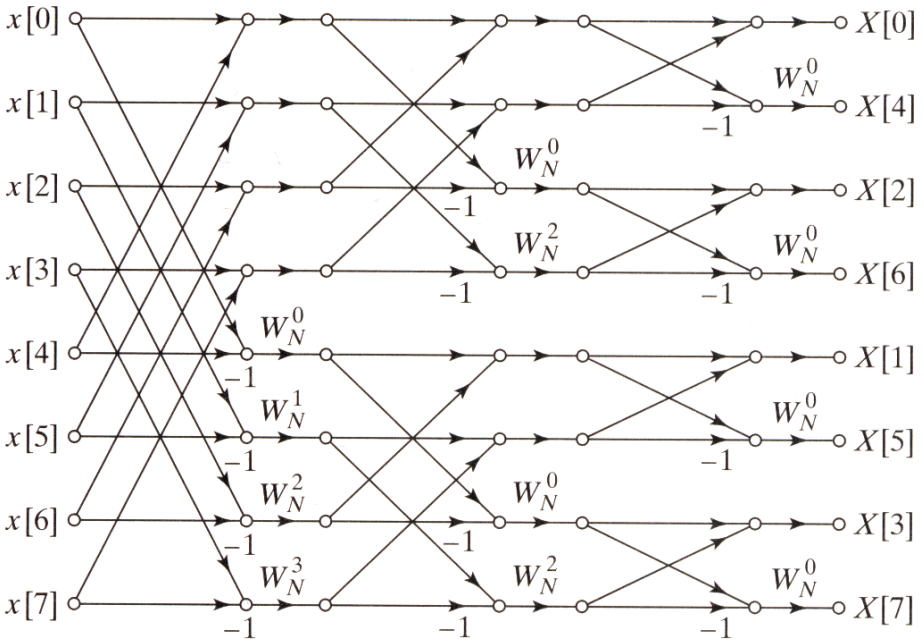


DIF FFT is a Transpose of DIT FFT

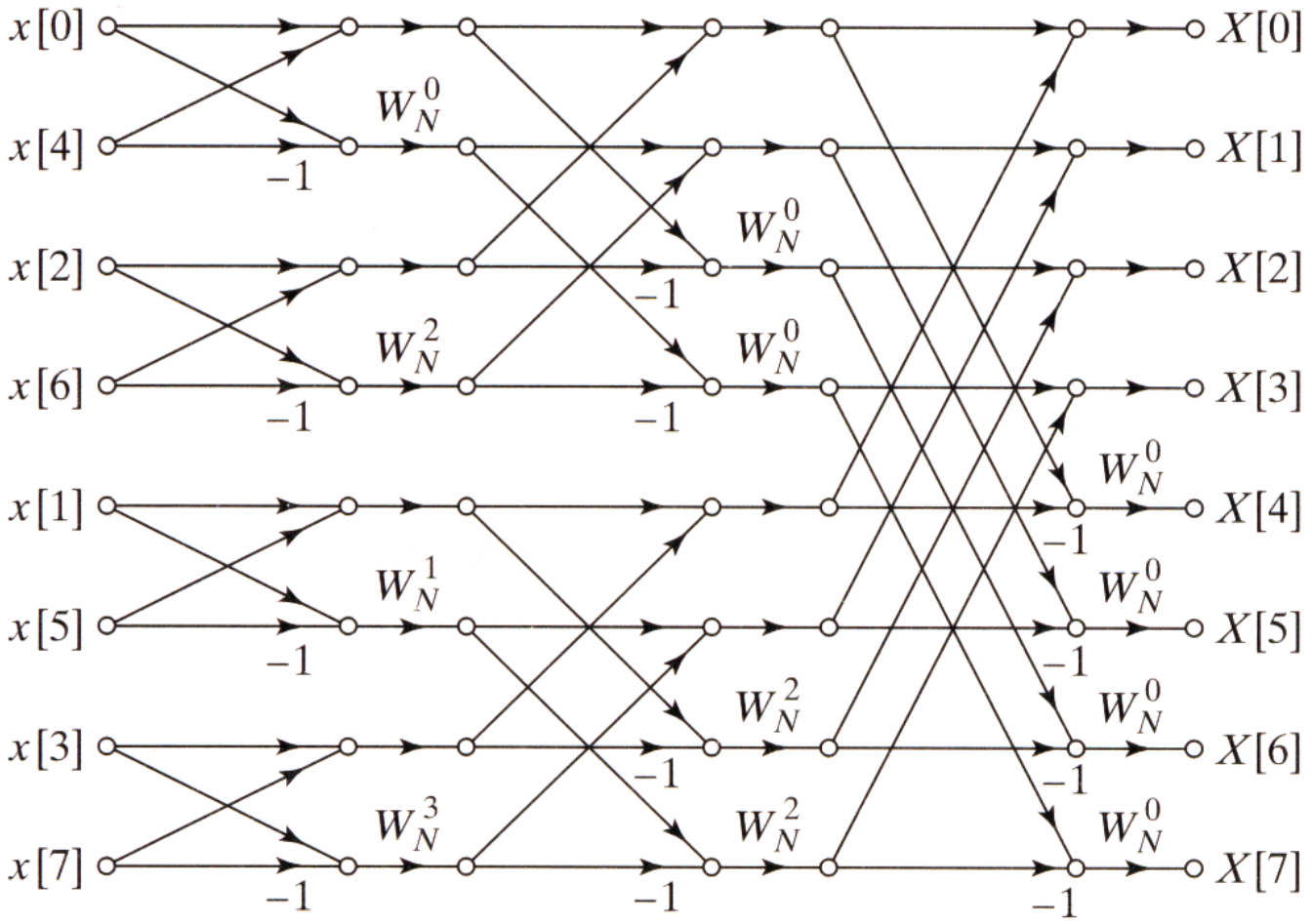
DIT FFT



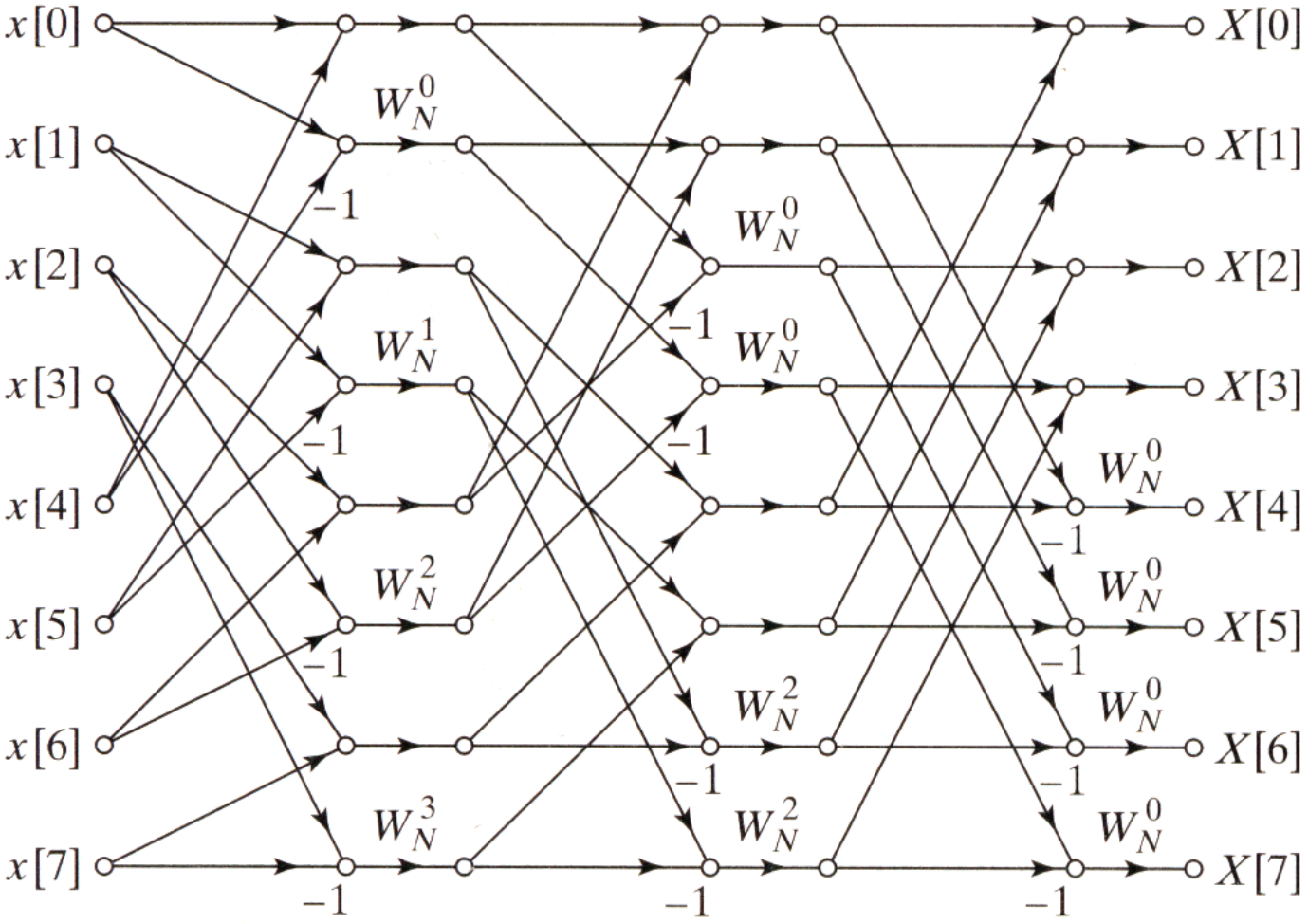
DIF FFT



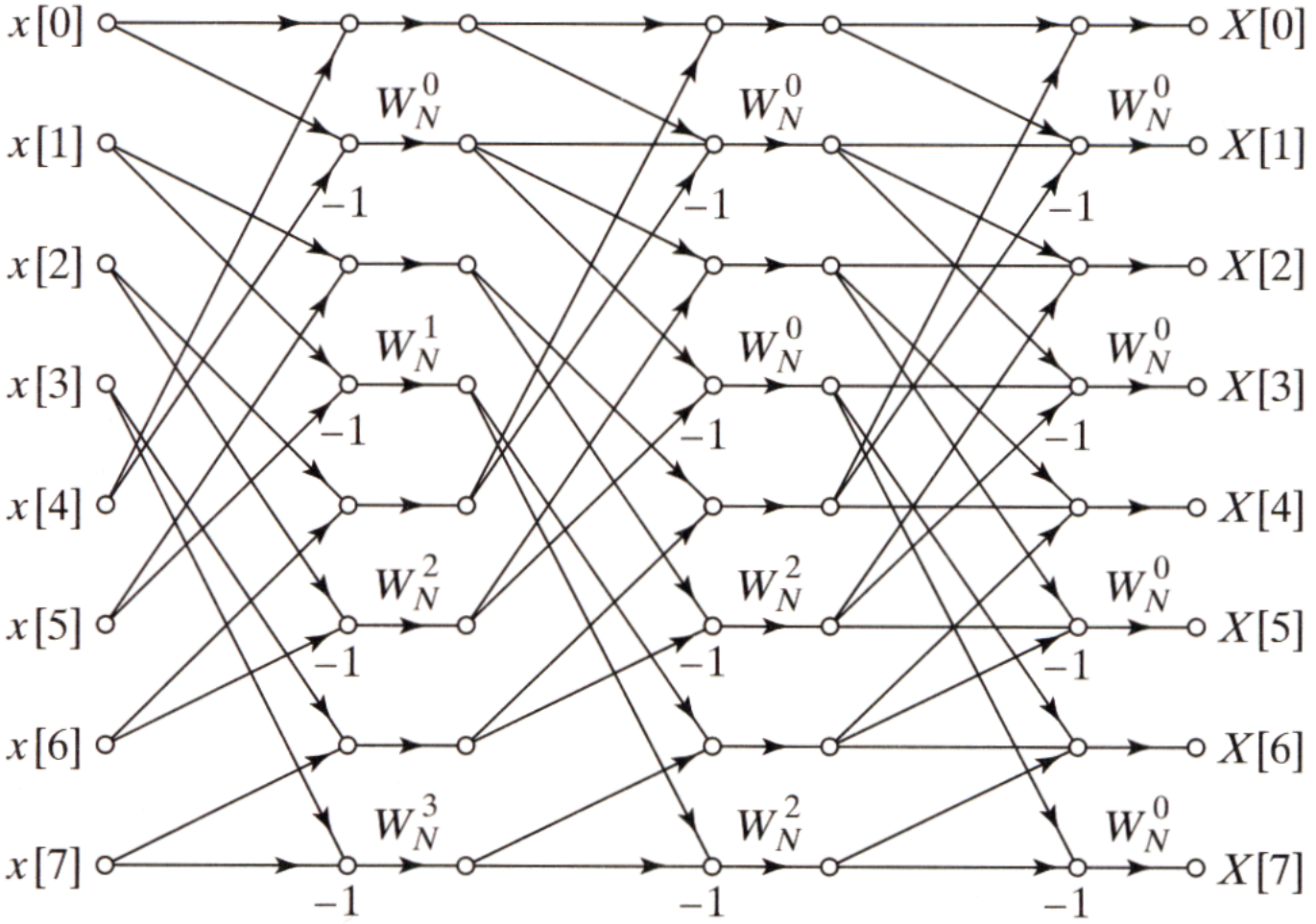
Alternative Forms (1/3)



Alternative Forms (2/3)



Alternative Forms (3/3)



Radix-4 FFT Algorithm

- The above FFT is radix-2
- Radix-4: decimation into 4 groups

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

$$=\sum_{n_2=0}^{(N/4)-1}x[4n_2+0]W_N^{(4n_2+0)k}+\sum_{n_2=0}^{(N/4)-1}x[4n_2+1]W_N^{(4n_2+1)k}$$

$$+\sum_{n_2=0}^{(N/4)-1}x[4n_2+2]W_N^{(4n_2+2)k}+\sum_{n_2=0}^{(N/4)-1}x[4n_2+3]W_N^{(4n_2+3)k}$$

$$=W_N^0\sum_{n_2=0}^{(N/4)-1}x[4n_2+0]W_{N/4}^{n_2k}+W_N^k\sum_{n_2=0}^{(N/4)-1}x[4n_2+1]W_{N/4}^{n_2k}$$

$$+W_N^{2k}\sum_{n_2=0}^{(N/4)-1}x[4n_2+2]W_{N/4}^{n_2k}+W_N^{3k}\sum_{n_2=0}^{(N/4)-1}x[4n_2+3]W_{N/4}^{n_2k}$$

$$=W_N^0F_0(k)+W_N^kF_1(k)+W_N^{2k}F_2(k)+W_N^{3k}F_3(k)$$

Radix-4 FFT Algorithm

$$X[k] = W_N^0 F_0(k) + W_N^k F_1(k) + W_N^{2k} F_2(k) + W_N^{3k} F_3(k)$$

$$X[k+(N/4)] = W_N^0 F_0(k) + W_N^{k+(N/4)} F_1(k) + W_N^{2[k+(N/4)]} F_2(k) + W_N^{3[k+(N/4)]} F_3(k)$$

$$= W_N^0 F_0(k) + W_N^{N/4} W_N^k F_1(k) + W_N^{N/2} W_N^{2k} F_2(k) + W_N^{3N/4} W_N^{3k} F_3(k)$$

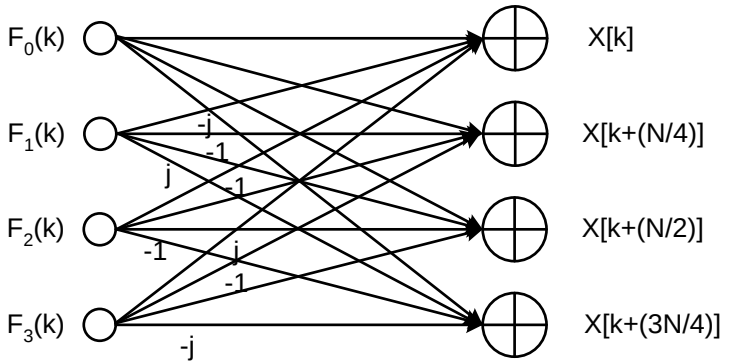
$$= F_0(k) - jW_N^k F_1(k) - W_N^{2k} F_2(k) + jW_N^{3k} F_3(k)$$

$$X[k+(N/2)] = F_0(k) - W_N^k F_1(k) + W_N^{2k} F_2(k) - W_N^{3k} F_3(k)$$

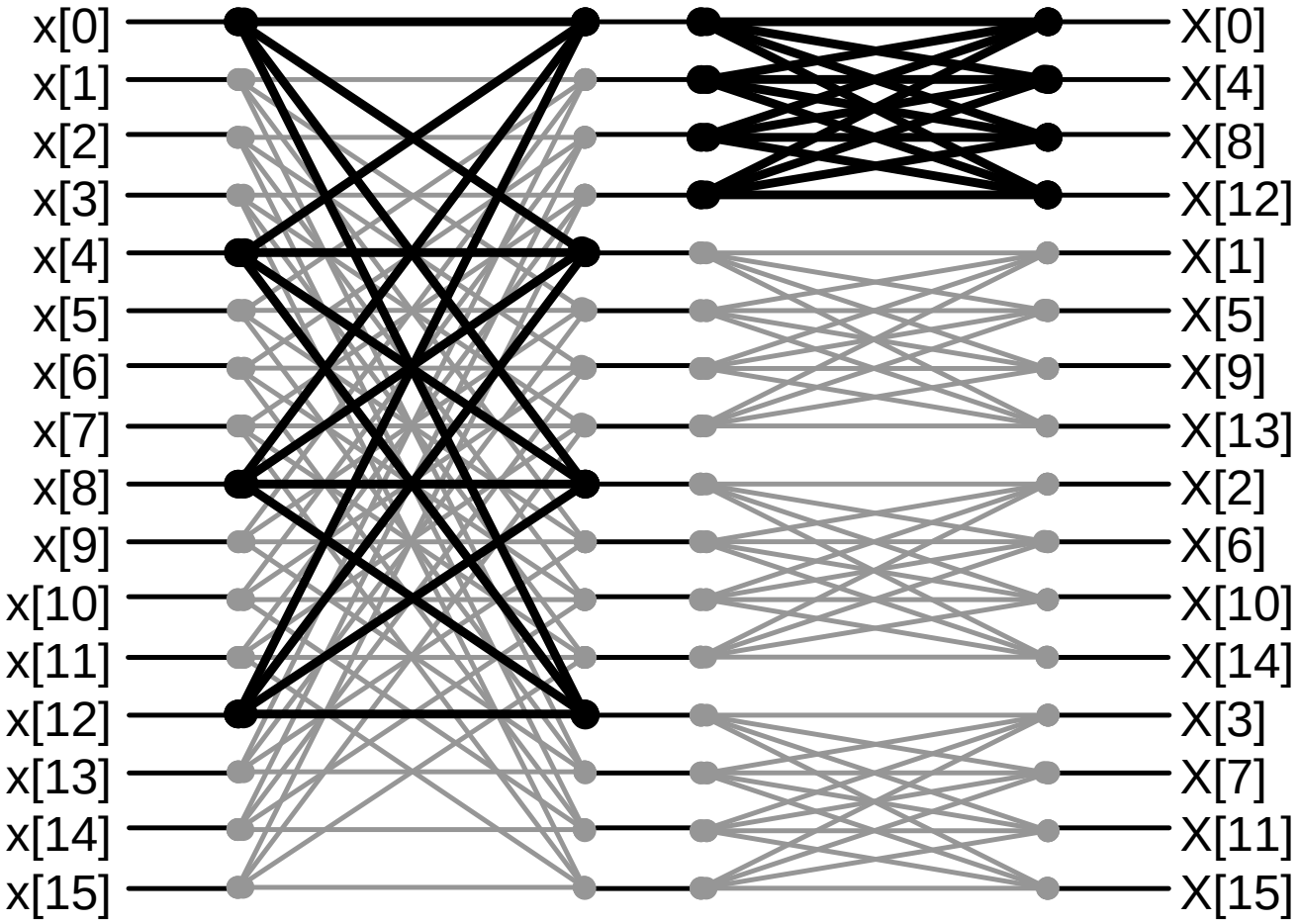
$$X[k+(3N/4)] = F_0(k) + jW_N^k F_1(k) - W_N^{2k} F_2(k) - jW_N^{3k} F_3(k)$$

Radix-4 butterfly matrix

$$\begin{bmatrix} X[k] \\ X[k+N/4] \\ X[k+N/2] \\ X[k+3N/4] \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} W_N^0 F_0(k) \\ W_N^k F_1(k) \\ W_N^{2k} F_2(k) \\ W_N^{3k} F_3(k) \end{bmatrix}$$



16-Point Radix-4 FFT



Radix-n FFT Algorithm

- For Radix-n FFT, the complexity is $N\log_n N$
- Larger n
  - Less stages
  - Less complex multiplier
  - More complex butterfly structure
- Radix-4 and radix-8 are suitable for long size FFT

Outline

- Introduction to FFT
- Algorithm of FFT
- Architectures of FFT

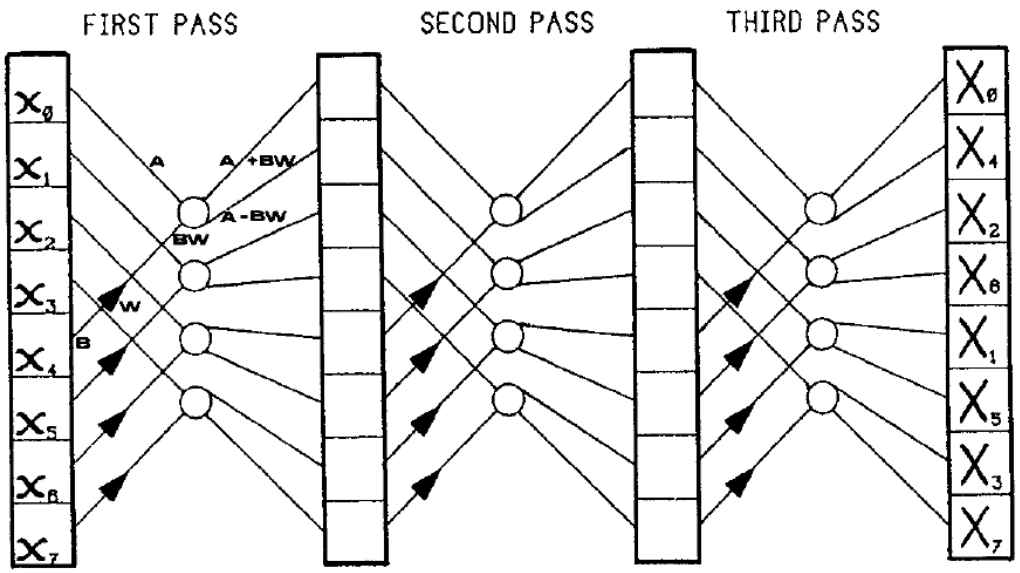
Distributed Arithmetic for FFT Butterfly

I. R. Mactaggart and M. A. Jack, “A single chip radix-2 FFT butterfly architecture using parallel data distributed arithmetic,” *IEEE Journal of Solid-State Circuits*, vol. sc-19, no. 3, June 1984.

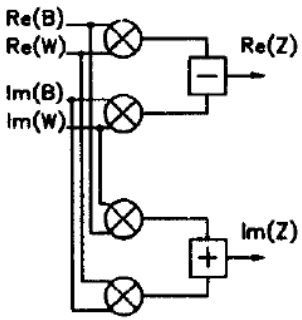
Introduction

- Reasons to apply DA for the butterfly computation
  - Low cost: accumulator + ROM
  - Highly regular VLSI architecture ➔ easy for layout

Basic Operation in FFT



A+**BW**



Z=BW

Re{Z}=Re{B} x Re{W} - Im{B} x Im{W}

Im{Z}=Re{B} x Im{W} + Re{W} x Im{B}

Map to DA (1/3)

- Take one of B or W as constant
  - For example, take B as constant
    - Re{Z}=Re{B} x Re{W} - Im{B} x Im{W}

TABLE II
DISTRIBUTED ARITHMETIC
ALGORITHM (COMPUTERS $Z = B \cdot W$)

W		REAL(Z)				IMAG(Z)			
Re	Im	1	2	3	4	5	6	7	8
0	0	0	−0	=K'−	K'	0	+0	=K−	K
0	1	0	−Im(B)	=K'−	K	Re(B)	+0	=K+	K'
1	0	Re(B)	−0	=K'+	K	0	+Im(B)	=K−	K'
1	1	Re(B)	−Im(B)	=K'+	K'	Re(B)	+Im(B)	=K+	K

(Bit-Level)

Where K=(Re(B)+Im(B))/2 and K'=(Re(B)−Im(B))/2

Map to DA (2/3)

$$\begin{aligned}\mathrm{Re}\{Z\} = {} & \overline{W}_{RO}\cdot\overline{W}_{IO}(0) \\ & + \overline{W}_{RO}\cdot W_{IO}(\mathrm{Im}\{B\}) \\ & + W_{RO}\cdot\overline{W}_{IO}(-\mathrm{Re}\{B\}) \\ & + W_{RO}\cdot W_{IO}(-\mathrm{Re}\{B\}+\mathrm{Im}\{B\}) \\ & + \sum_{n=1}^{N-1}\Big[\overline{W}_{Rn}\cdot\overline{W}_{In}(0) \\ & \quad + \overline{W}_{Rn}\cdot W_{In}(-\mathrm{Im}\{B\}) \\ & \quad + W_{Rn}\cdot\overline{W}_{In}(\mathrm{Re}\{B\}) \\ & \quad + W_{Rn}\cdot W_{In}(\mathrm{Re}\{B\}-\mathrm{Im}\{B\})\Big]\cdot 2^{-n}.\end{aligned}$$

$$K = (\mathrm{Re}\{B\}+\mathrm{Im}\{B\})/2 \quad \text{and}$$

$$K' = (\mathrm{Re}\{B\}-\mathrm{Im}\{B\})/2.$$

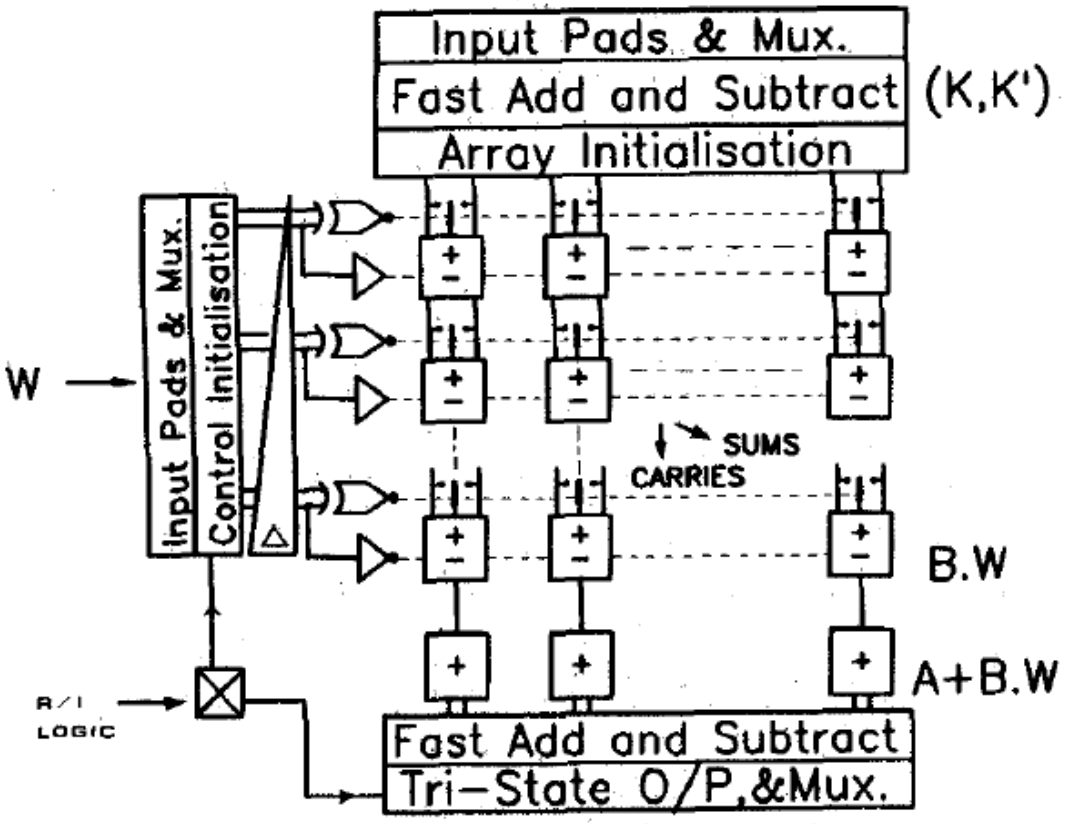
$$\begin{aligned}\mathrm{Re}\{Z\} = {} & \overline{W}_{RO}\cdot\overline{W}_{IO}(-K'+K') \\ & + \overline{W}_{RO}W_{IO}(-K'+K) \\ & + W_{RO}\overline{W}_{IO}(-K'-K) \\ & + W_{RO}W_{IO}(-K'-K') \\ & + \sum_{n=1}^{N-1}\Big[\overline{W}_{Rn}\cdot\overline{W}_{In}(K'-K') \\ & \quad \overline{W}_{Rn}\cdot W_{In}(K'-K) \\ & \quad W_{Rn}\cdot\overline{W}_{In}(K'+K) \\ & \quad W_{Rn}\cdot W_{In}(K'+K')\Big]\cdot 2^{-n}\end{aligned}$$

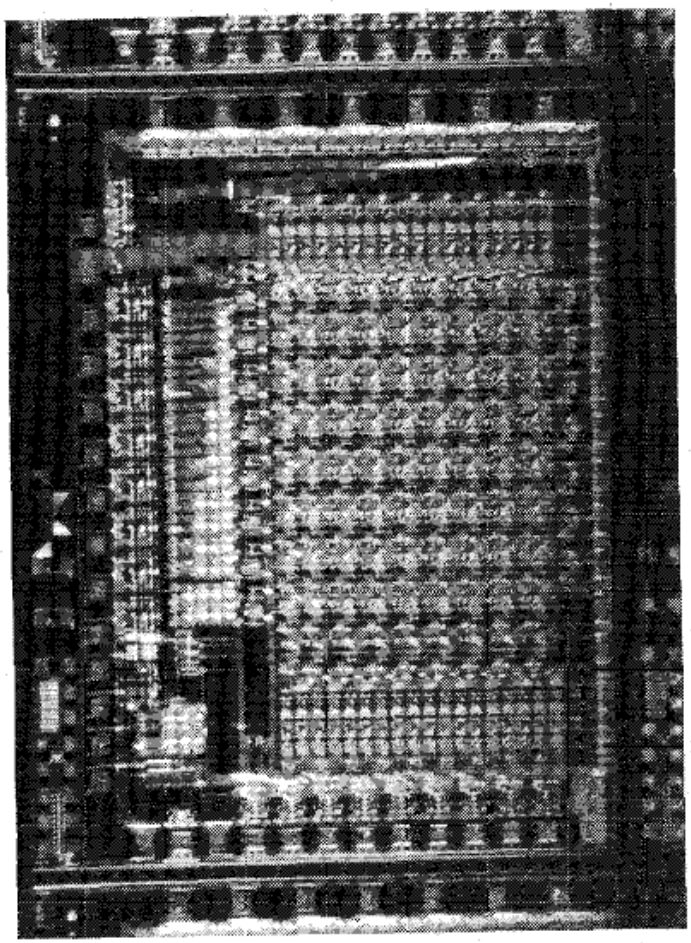
Map to DA (3/3)

$$\begin{aligned}\mathrm{Re}\{Z\} = {} & \overline{W}_{RO}\cdot\overline{W}_{IO}(-K'+K') \\ & + \overline{W}_{RO}W_{IO}(-K'+K) \\ & + W_{RO}\overline{W}_{IO}(-K'-K) \\ & + W_{RO}W_{IO}(-K'-K') \\ & + \sum_{n=1}^{N-1}\Big[\overline{W}_{Rn}\cdot\overline{W}_{In}(K'-K') \\ & \quad \overline{W}_{Rn}\cdot W_{In}(K'-K) \\ & \quad W_{Rn}\cdot\overline{W}_{In}(K'+K) \\ & \quad W_{Rn}\cdot W_{In}(K'+K')\Big]\cdot 2^{-n}\end{aligned}$$

$$\begin{aligned}\mathrm{Re}\{Z\} = {} & -K'\cdot 2^{-(N-1)} + \overline{W}_{RO}\cdot\overline{W}_{IO}(+K') \\ & + \overline{W}_{RO}\cdot W_{IO}(+K) \\ & + W_{RO}\cdot\overline{W}_{IO}(-K) \\ & + W_{RO}\cdot W_{IO}(-K') \\ & + \sum_{n=1}^{N-1}\Big[+\overline{W}_{Rn}\cdot\overline{W}_{In}(-K') \\ & \quad + \overline{W}_{Rn}\cdot W_{In}(-K) \\ & \quad + W_{Rn}\cdot\overline{W}_{In}(+K) \\ & \quad + W_{Rn}\cdot W_{In}(+K')\Big]\cdot 2^{-n}.\end{aligned}$$

Architecture and Chip Photo





2D Array Architecture

M.-K. Lee, K.-W. Shih, and J.-K. Lee, “A VLSI array processor for 16-point FFT,” *IEEE Journal of Solid-State Circuits*, vol. 26, no. 9, Sept. 1991.

Comparison Between Different Architectures

Not suitable for VLSI

That's why we should use array architecture

TABLE I
COMPARISON OF VARIOUS PARALLEL ARCHITECTURES FOR FFT COMPUTATION

		FFT network [7]	Perfect shuffle network [8]	2-D mesh array [9]–[12]	Parallel arch. using DC's [13]	Parallel arch. using SE's [14]	1-D systolic DFT array [15]
Number of PE's		$(N/2)\log N$	$N/2$	N	$\log N$	$N \log N$	$2N-1$
Butterfly steps		pipelined	$\log N$	$\log N$	pipelined	pipelined	pipelined
Interconnection area		$O(N)$	$O(N^2/\log^2 N)$	No area	No area	No area	No area
Data shuffling time		$O(\log N)$	$O(\log N)$	$O(\sqrt{N})$	$O(N)$	$O(N)$	$O(N)$
Interconnection regularity		irregular	irregular	very regular	regular	regular	very regular
Additional hardware for data shuffling		No	No	No	$O(N \log N)$ DC's	$O(N \log N)$ SE's	No
Dominant factor	Time	Pipelined	Butterfly steps	Data shuffling	pipelined	pipelined	pipelined
	Area	interconnection	interconnection	PE's	PE's + DC's	PE's + SE's	PE's

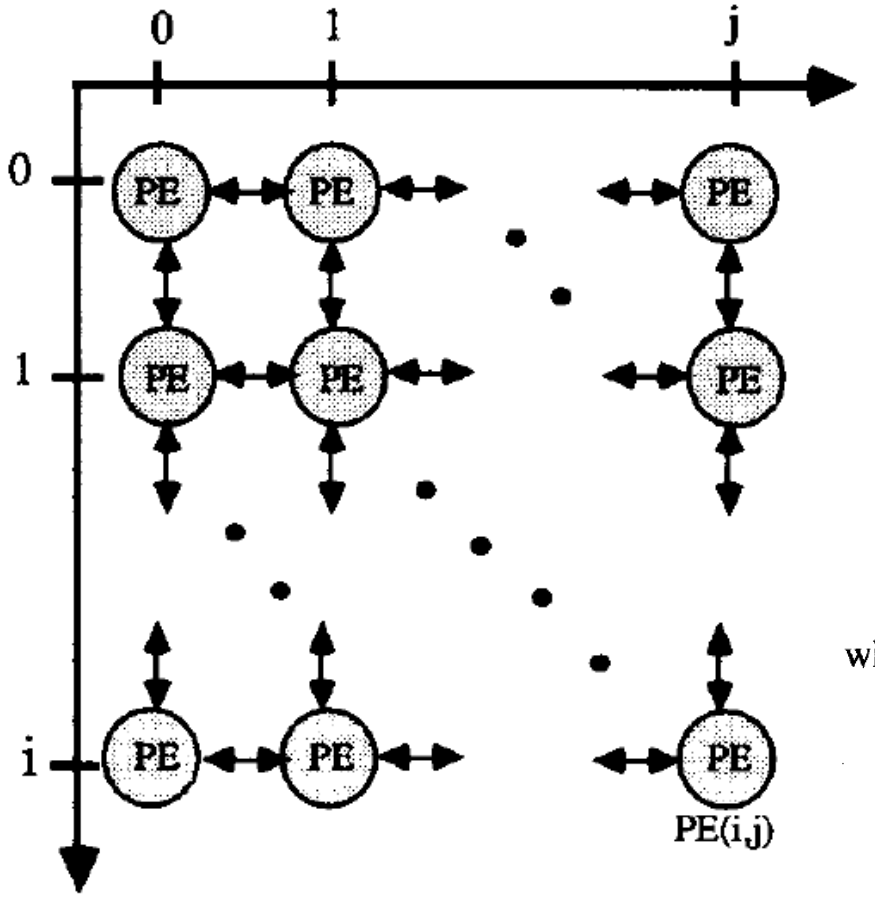
PE's: Processing Elements, DC's: Delay Commutators, SE's: Systolic Elevators

However…

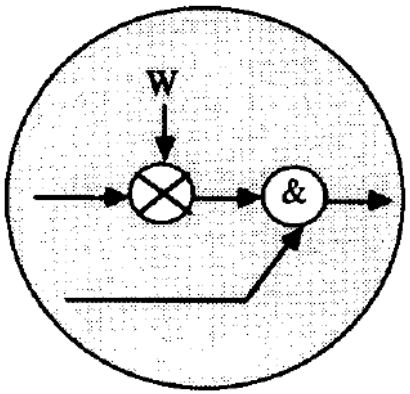
2-D Mesh Array

- For N-point FFT, use M butterfly PEs
  - If (2M<=N)
    - Require P=N/M storages for each PE
    - Not so regular for data access, need $\log_2 N$ bits for data addressing
  - If (M=N)
    - Require data reshuffling operation after butterfly
    - Only half of the PEs in the array participate in butterfly computation
- Proposed architecture
  - M=N, adopt half-butterfly arithmetic (HBA) as the arithmetic function of PE rather than radix-2 butterfly

2-D Mesh Array and HBA



HBA



operator '&' denotes '+' or '-'

$$\text{HBA\&} = f(q-1,k)\,\&\,f(q-1,k+D(q))\cdot W^{pt}$$

where

$$q = 1,2,3,\cdots,\log_2 N$$

$$W^{pt} = \exp(-j2\pi pt/N).$$

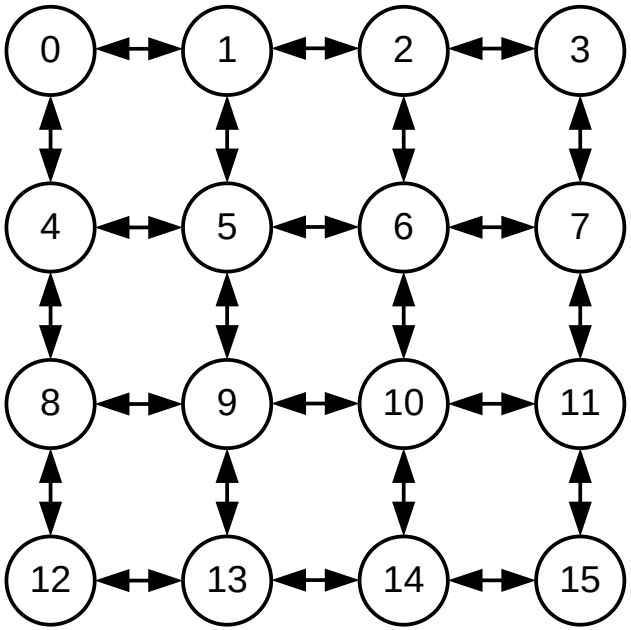
q: stage

Procedure for Computing a 16-Point FFT (1/6)

- Begin
  - For q=1 to 4 do in parallel for all PEs
    - Begin
      - Data shuffling
      - HBA computation
    - END
- End

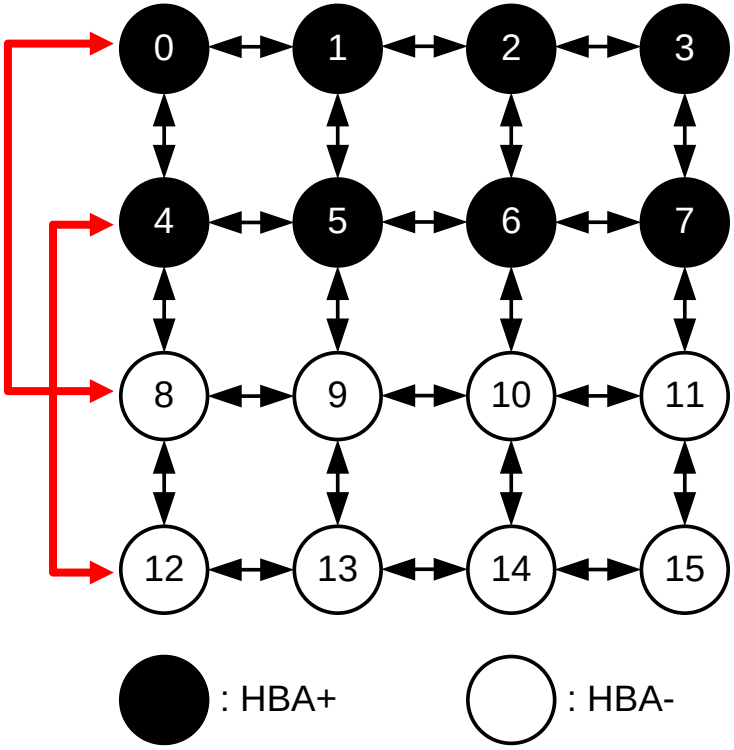
Procedure for Computing a 16-Point FFT (2/6)

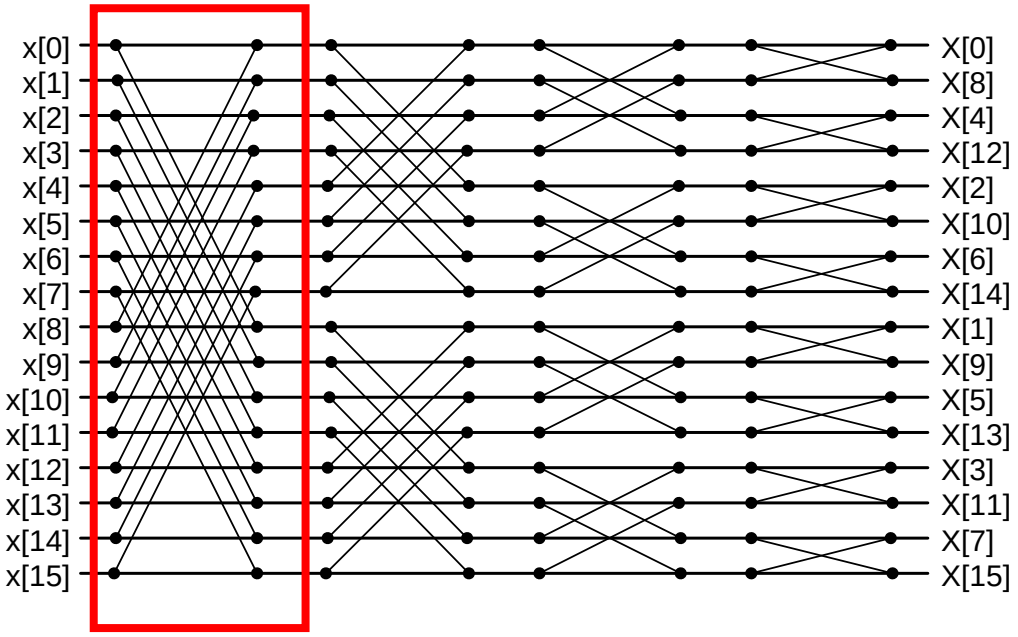
- Initially, 16 input data are loaded into the array in row-major order



Procedure for Computing a 16-Point FFT (3/6)

- Stage1

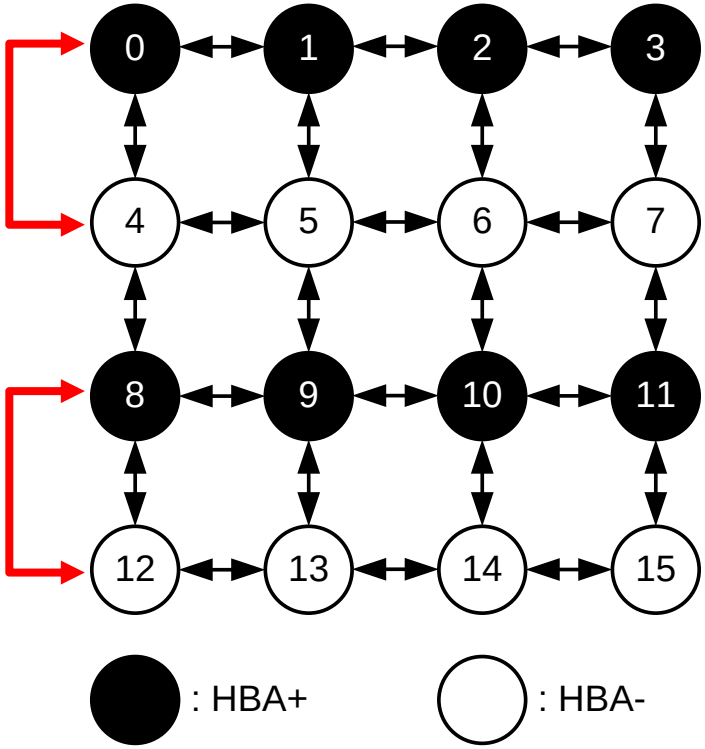


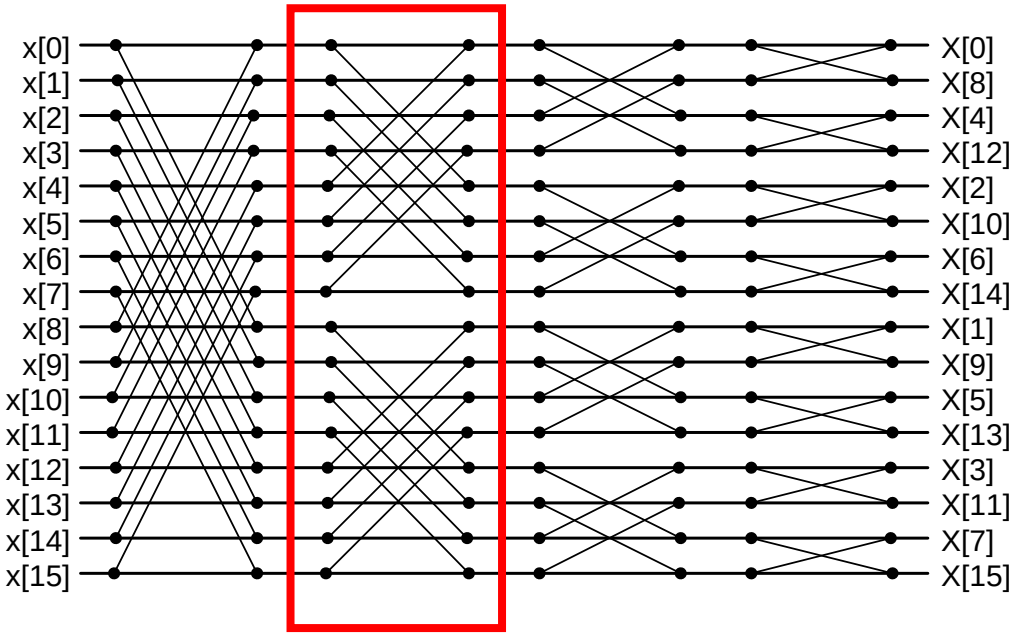


Note: the data communication take 2 cycles to cross 2 PEs

Procedure for Computing a 16-Point FFT (4/6)

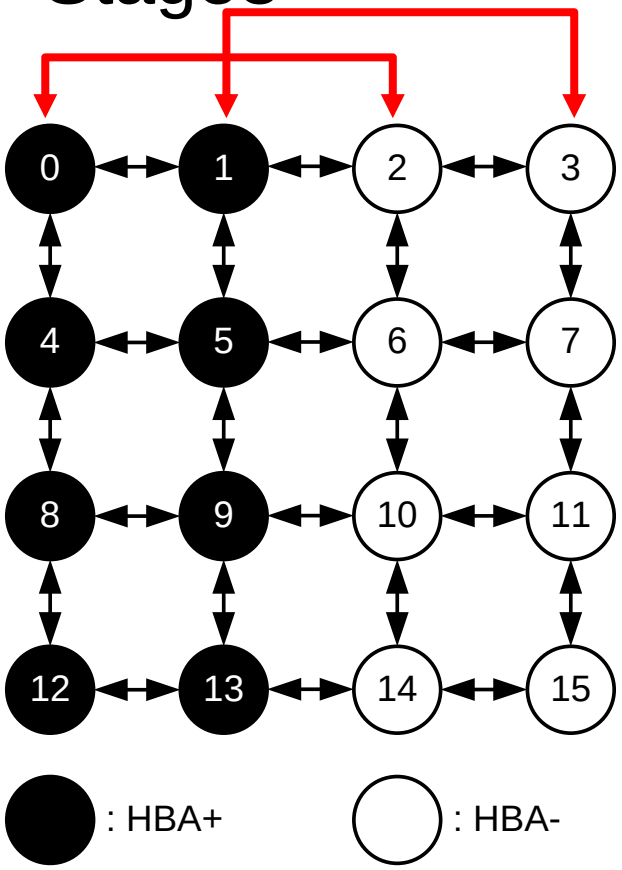
- Stage2

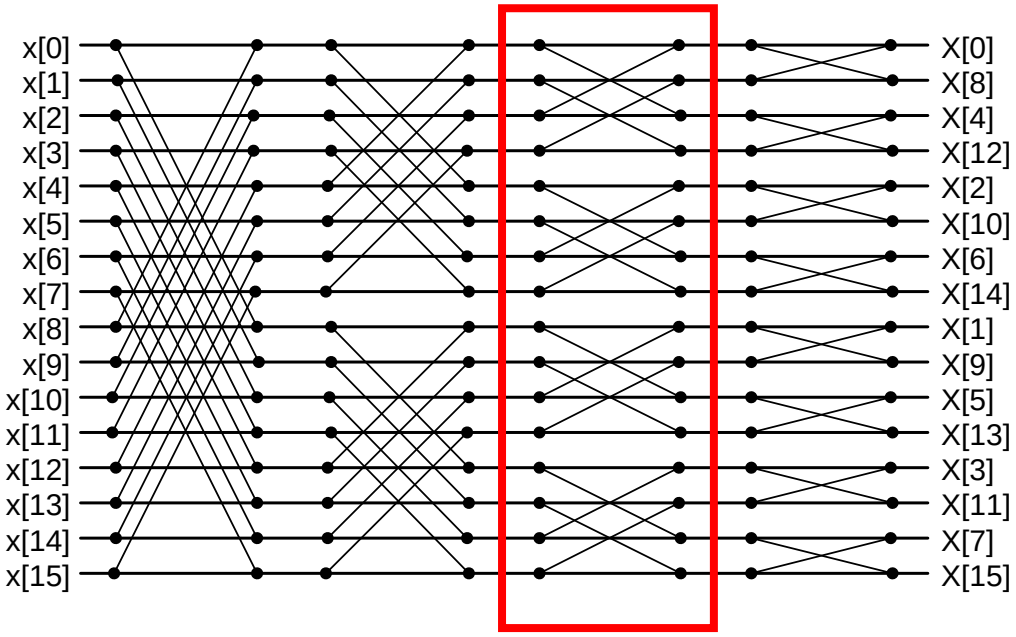




Procedure for Computing a 16-Point FFT (5/6)

- Stage3

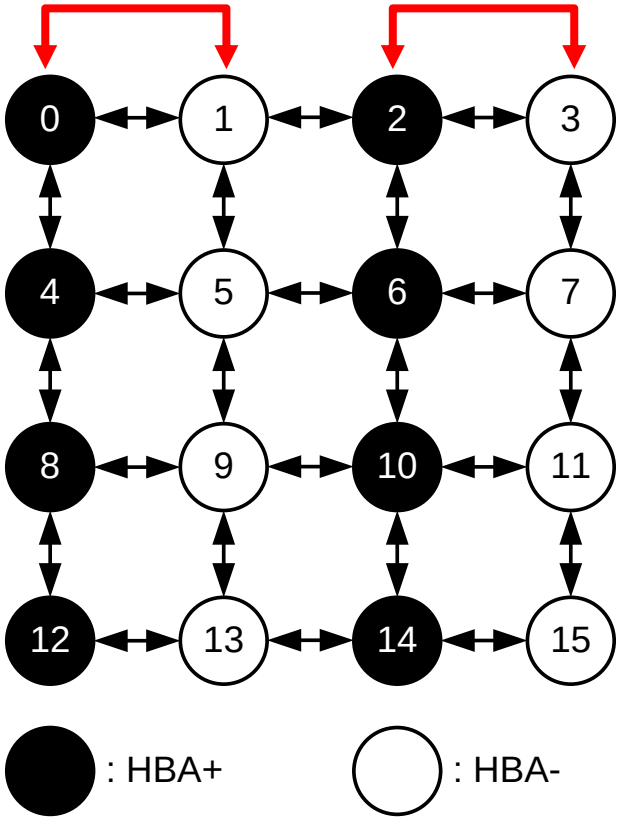


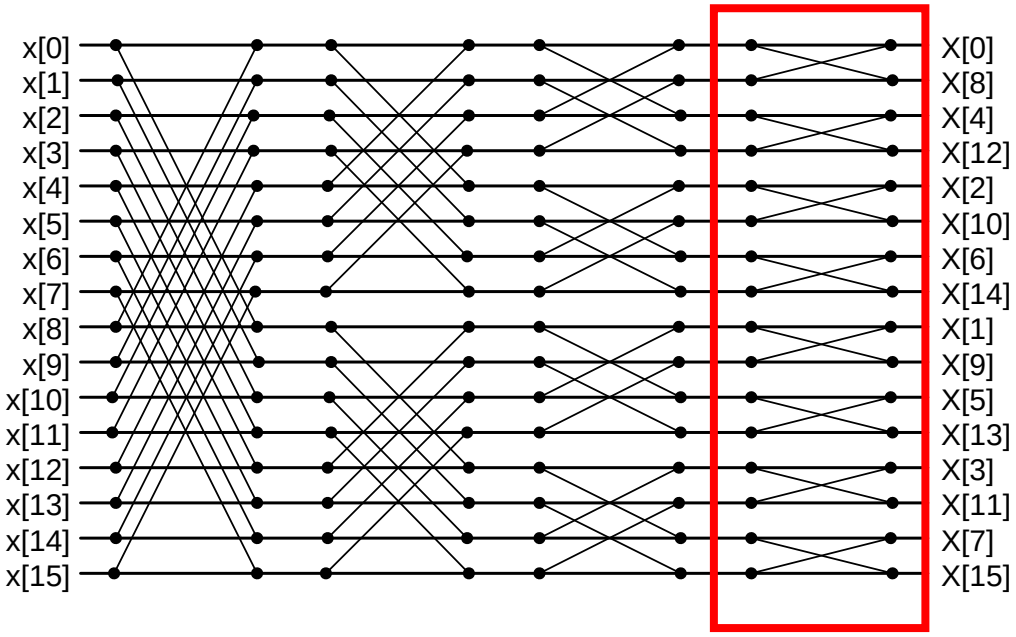


Note: the data communication take 2 cycles to cross 2 PEs

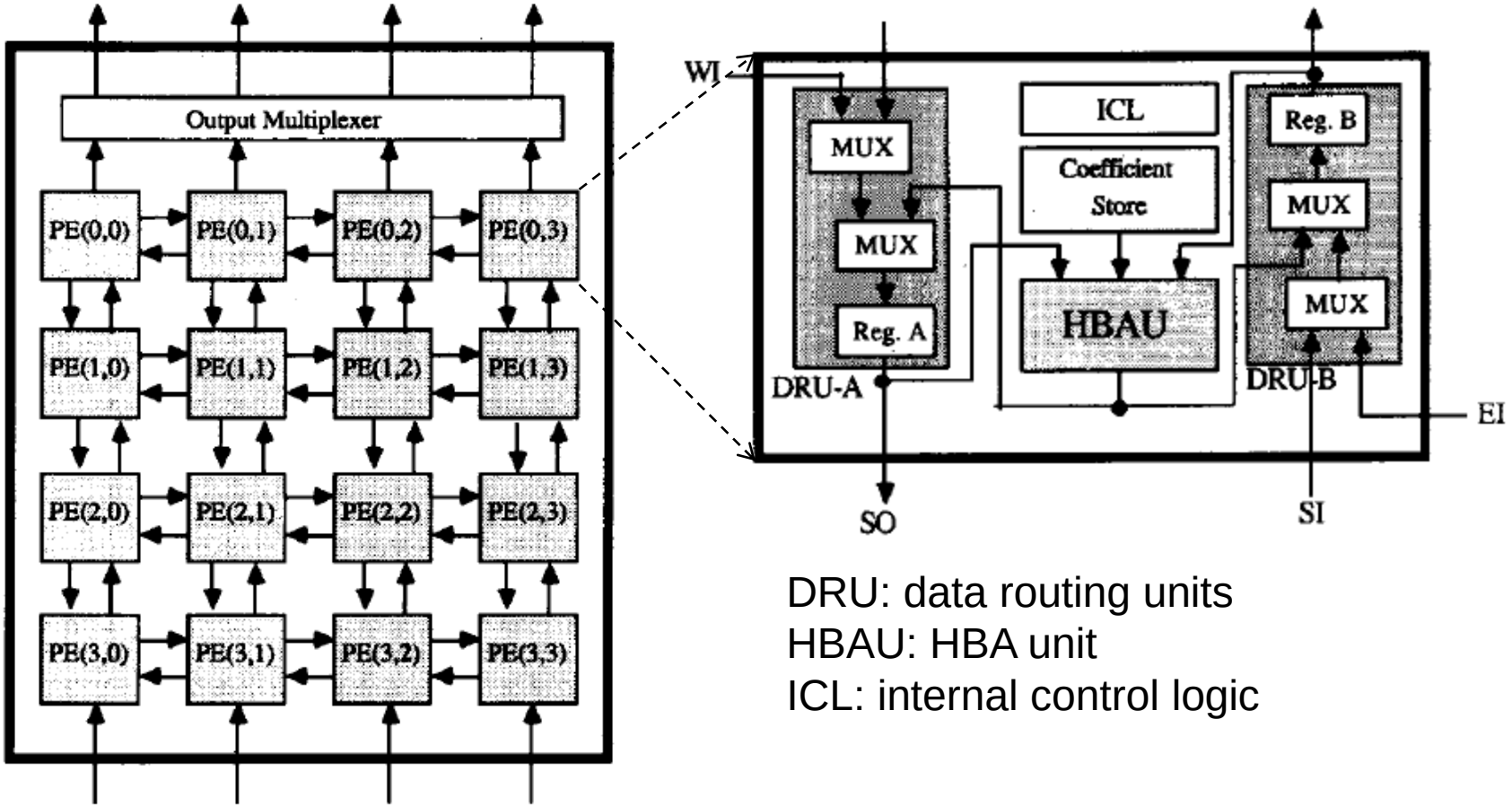
Procedure for Computing a 16-Point FFT (6/6)

- Stage4



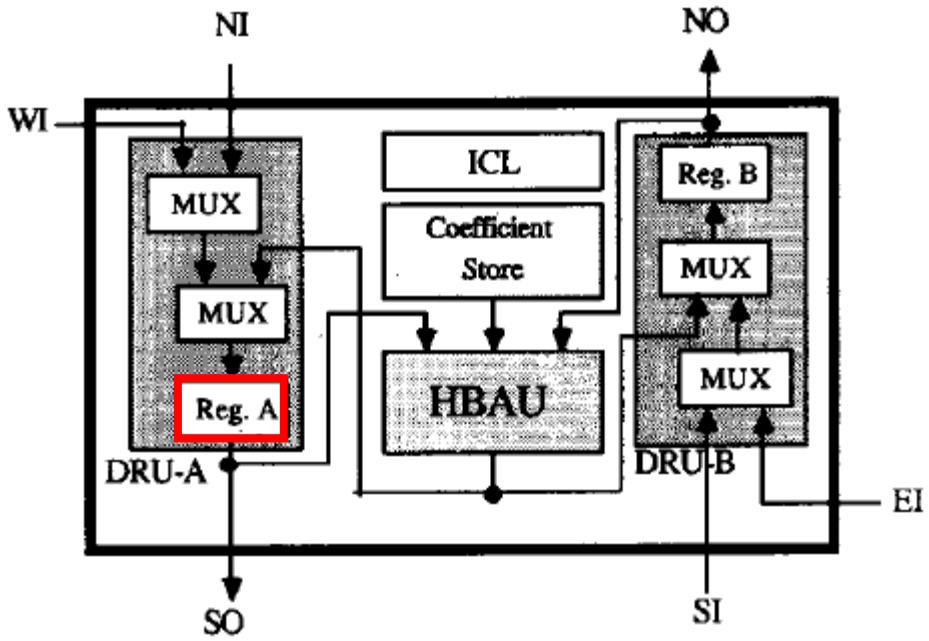


Chip Architecture



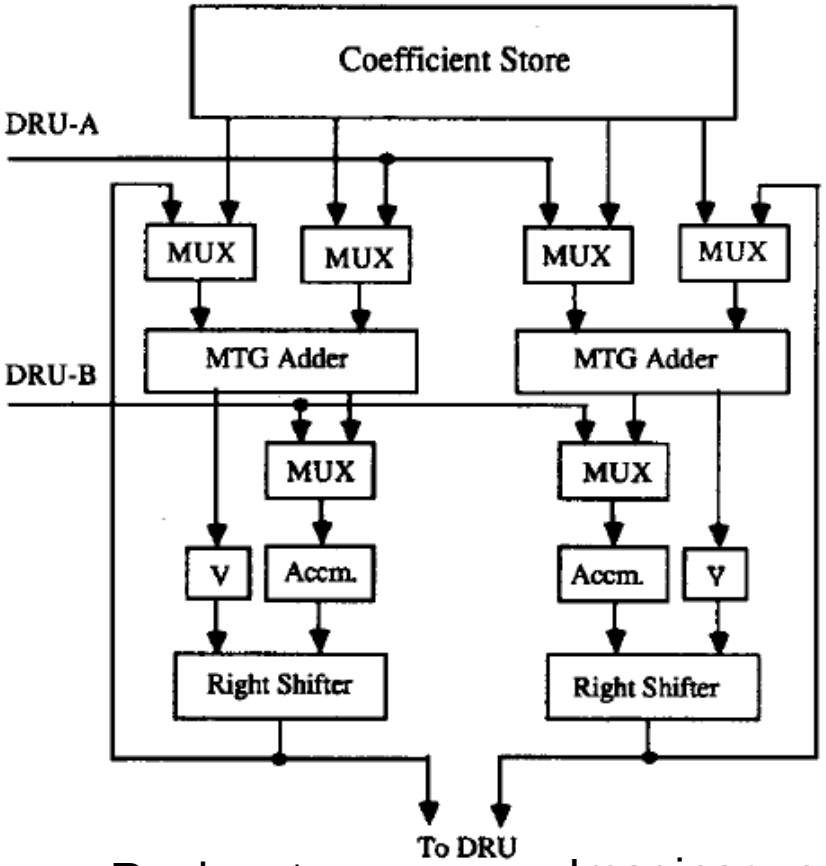
DRU: data routing units
HBAU: HBA unit
ICL: internal control logic

Chip Architecture



- Configurable register
  - PIPO (parallel-in, parallel-out) register for data communication
  - PISO (parallel-in, serial-out) register for HBA computation

HBAU Architecture



- Use distributed arithmetic
- MTG adder: modified transmission gate adder

Layout Statistics

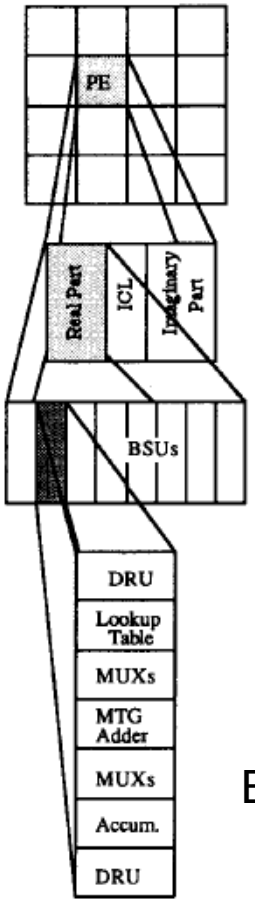


TABLE II
LAYOUT STATISTICS

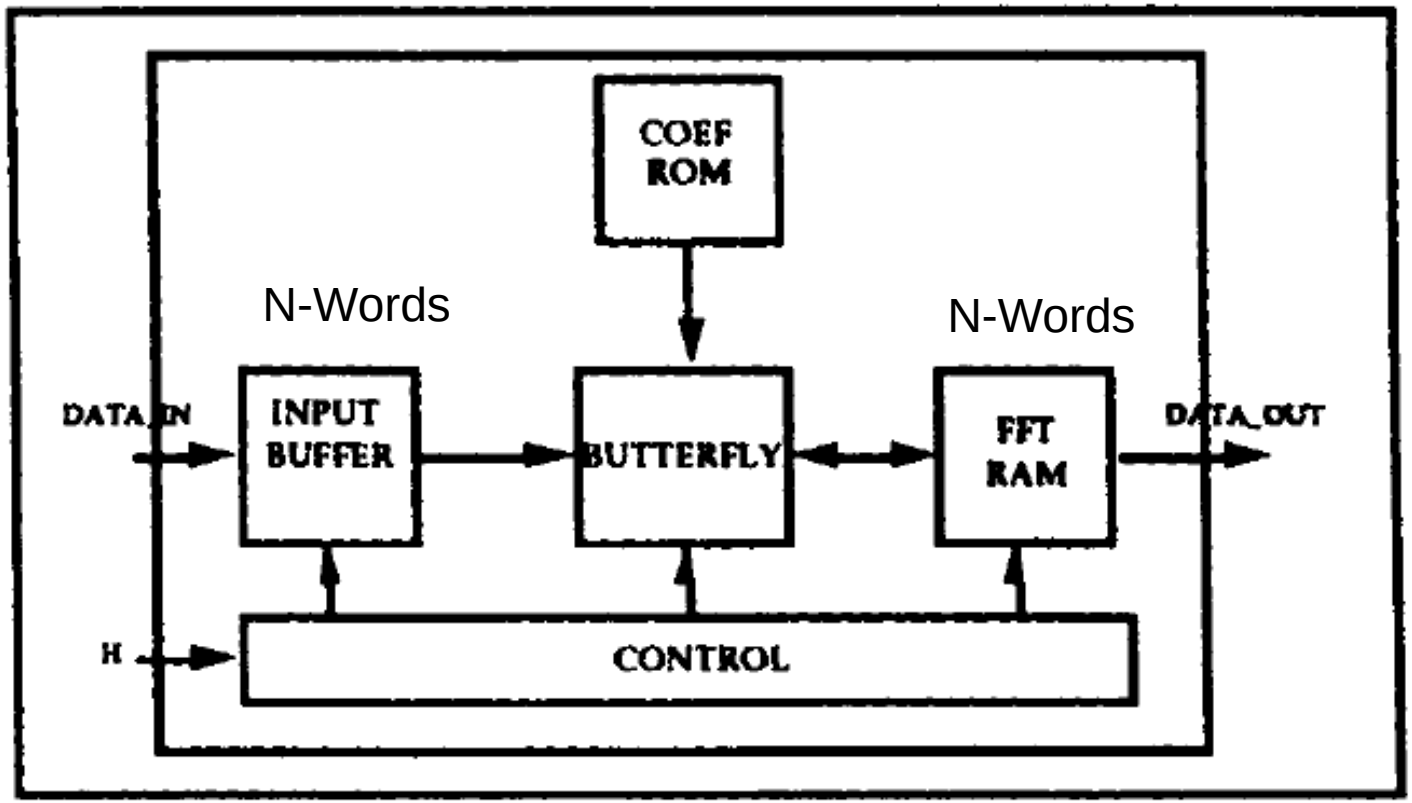
Block	Area (mm^2)	Transistors
BSU	0.132	150
ICL	0.967	526
PE	3.345	2998
4×4 Array	53.52	48 000
Die size including 83 pads		76.56 mm^2

BSU: bit-slice unit

Pipelined FFT Processors

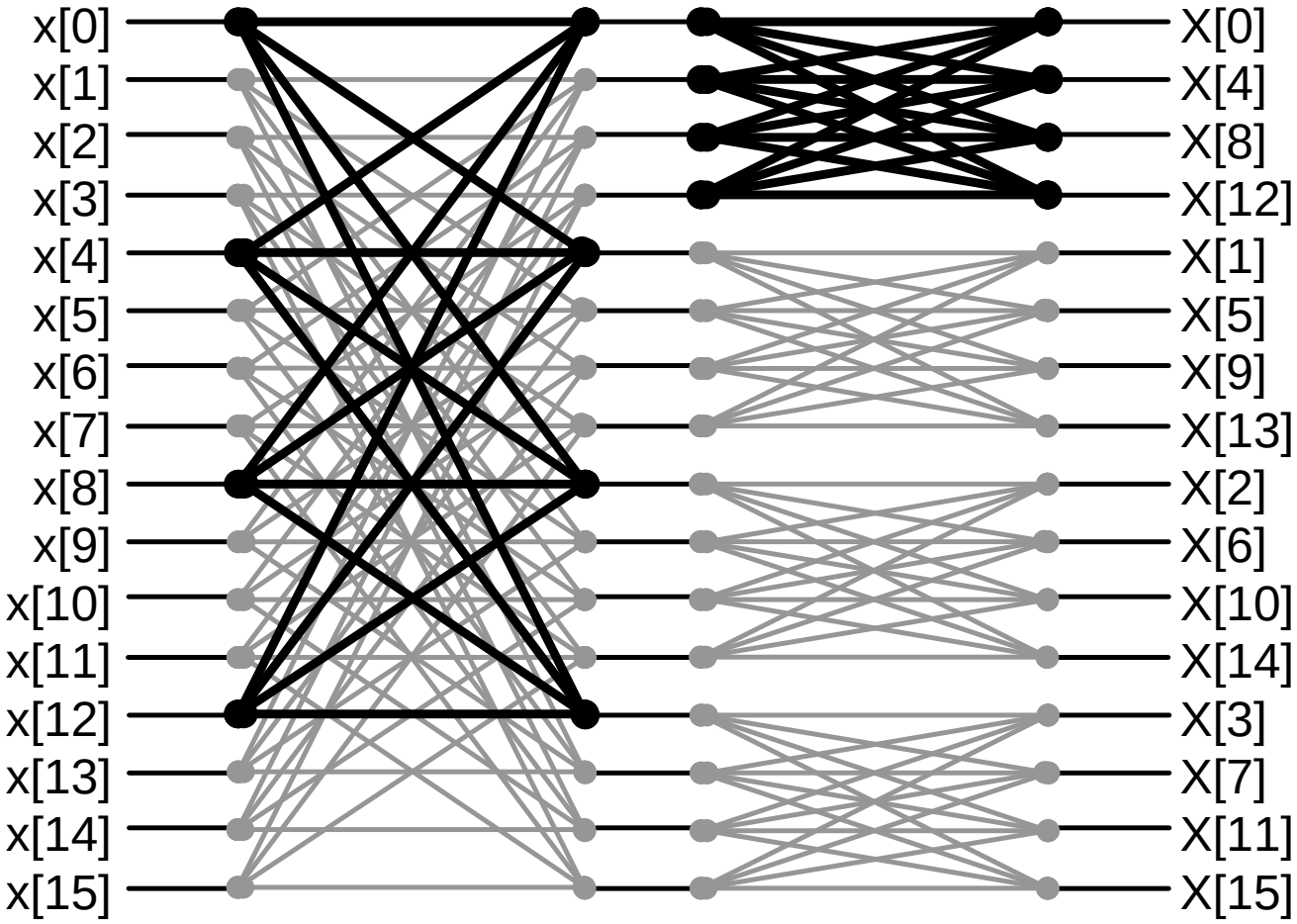
1. E. E. Swartzlander, W. K. W. Young, and S. J. Joseph, “A radix 4 delay commutator for fast Fourier transform processor implementation,” *IEEE Journal of Solid-State Circuits*, vol. sc-19, no. 5, Oct. 1984.
2. S. He and M. Torkelson, “Design and implementation of a 1024-point pipeline FFT processor,” *IEEE Custom Integrated Circuits Conference*, 1998.
3. E. Bidet, D. Castelain, C. Joanblanq, and P. Seen, “A fast single-chip implementation of 8192 complex point FFT,” *IEEE Journal of Solid-State Circuits*, vol. 30, no. 3, March 1995.
4. Y.-N. Chang and K. K. Parhi, “An efficient pipelined FFT architecture,” *IEEE Transactions on Circuits and Systems---II: Analog and Digital Signal Processing*, vol. 50, no. 6, June 2003.

Conventional FFT Implementation

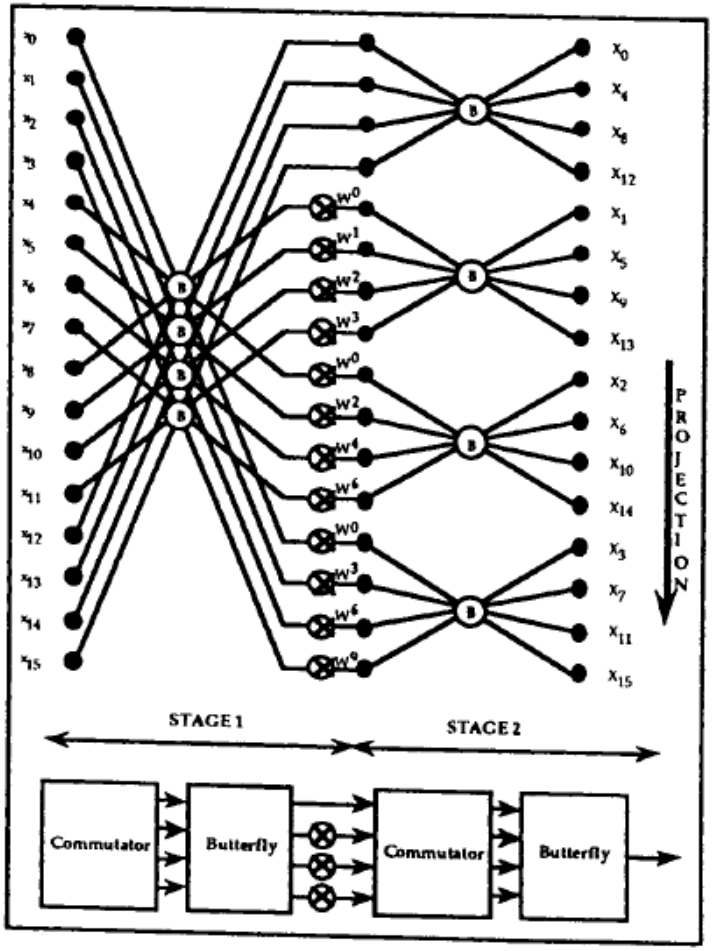


For in-place operation, N Words are enough, instead of 2N words.

16-Point Radix-4 FFT

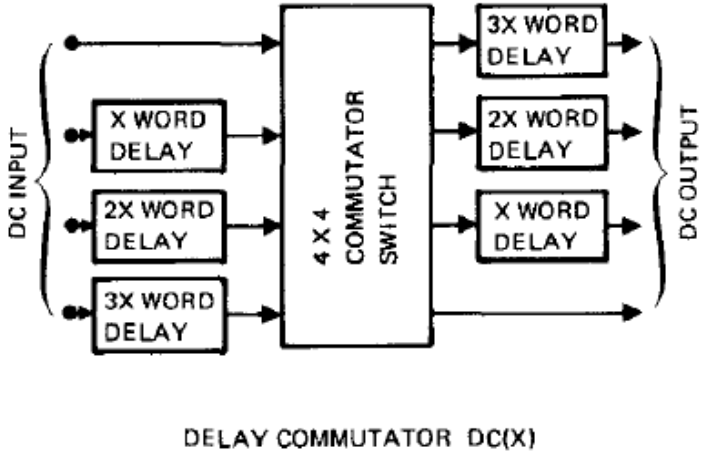


Pipeline FFT Architecture



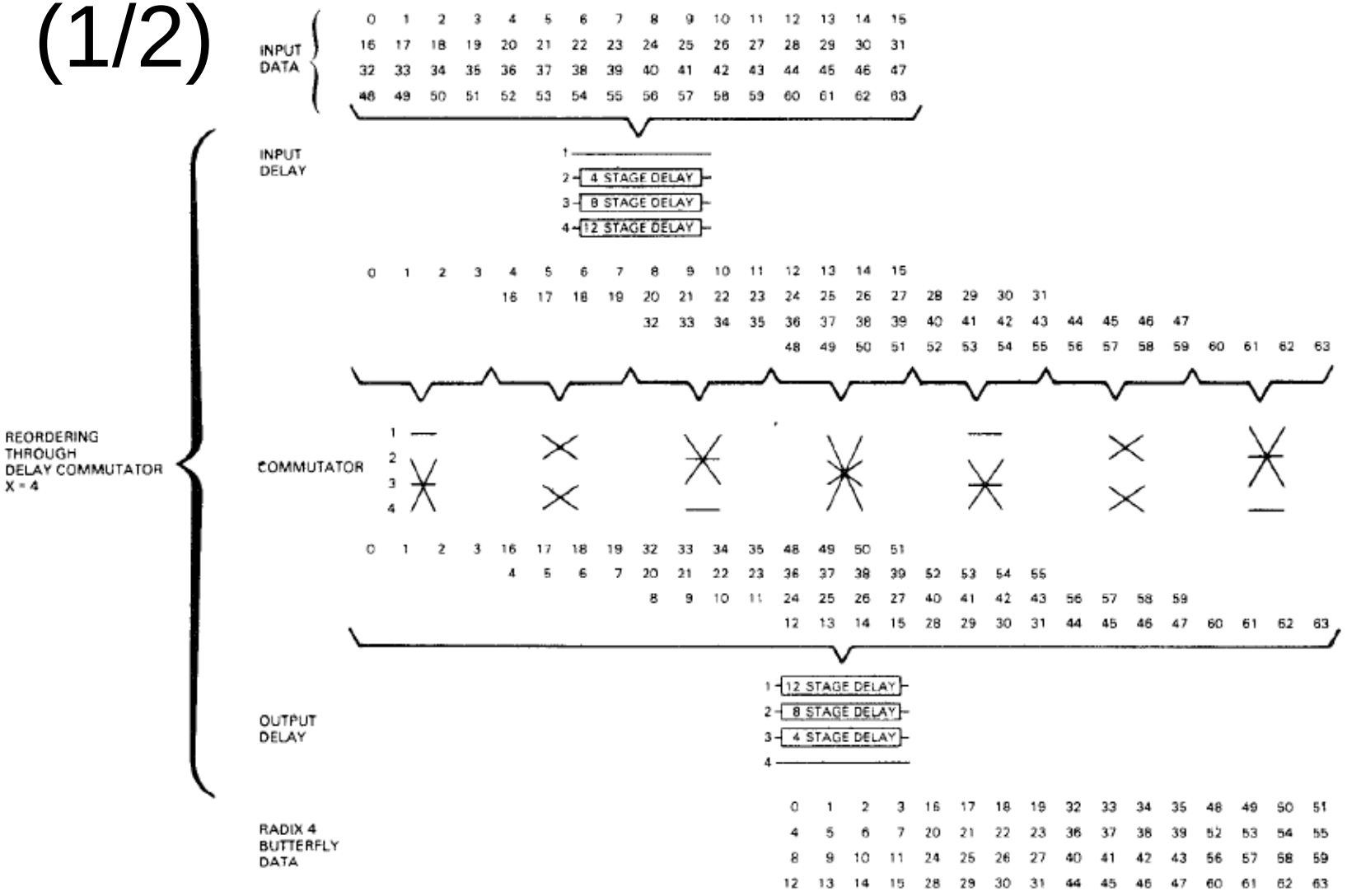
DC(4) DC(1)

- Projected into two data commutators and two butterflies

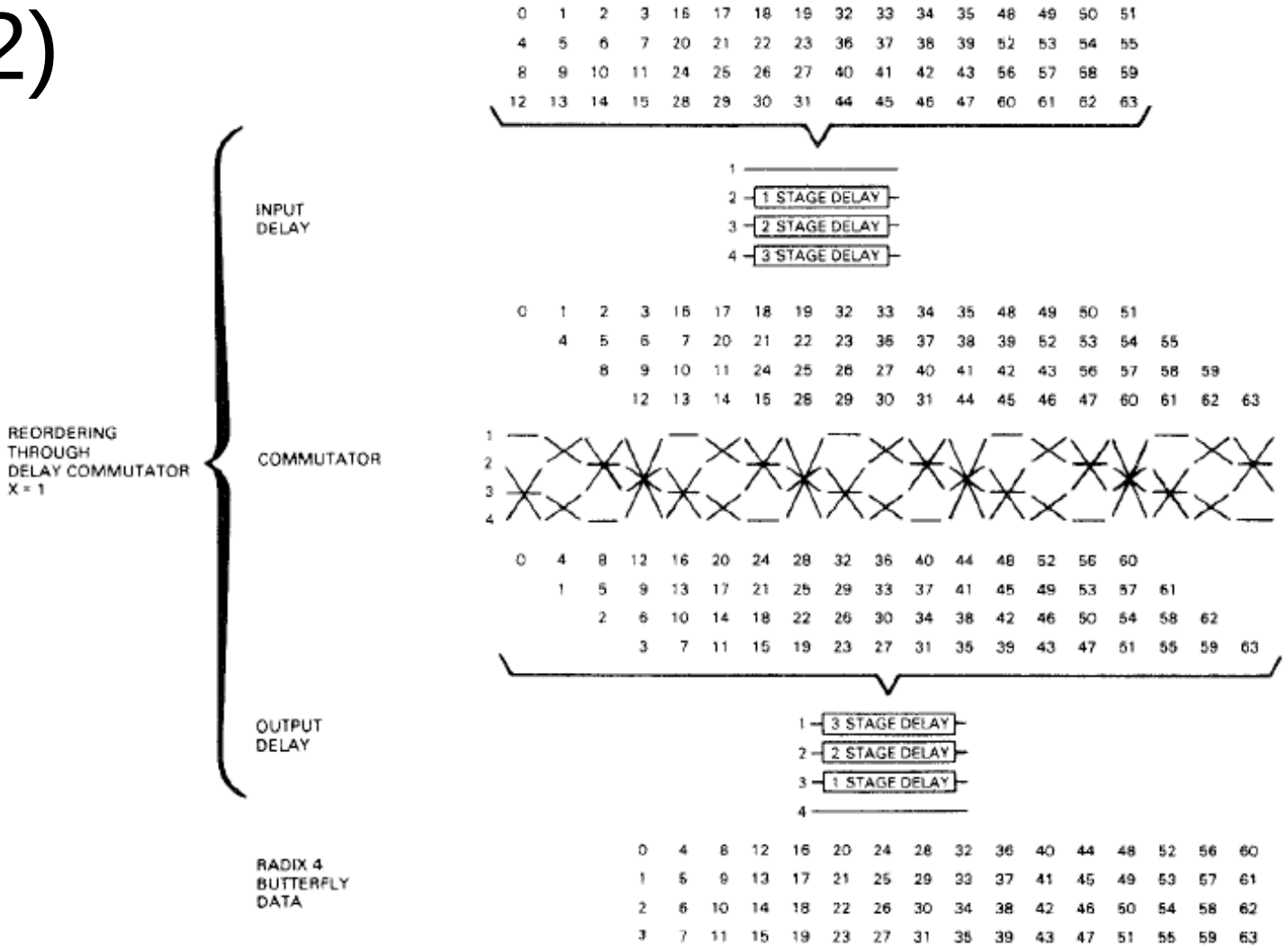


DELAY COMMUTATOR DC(X)

How does the Commutator Work? (1/2)

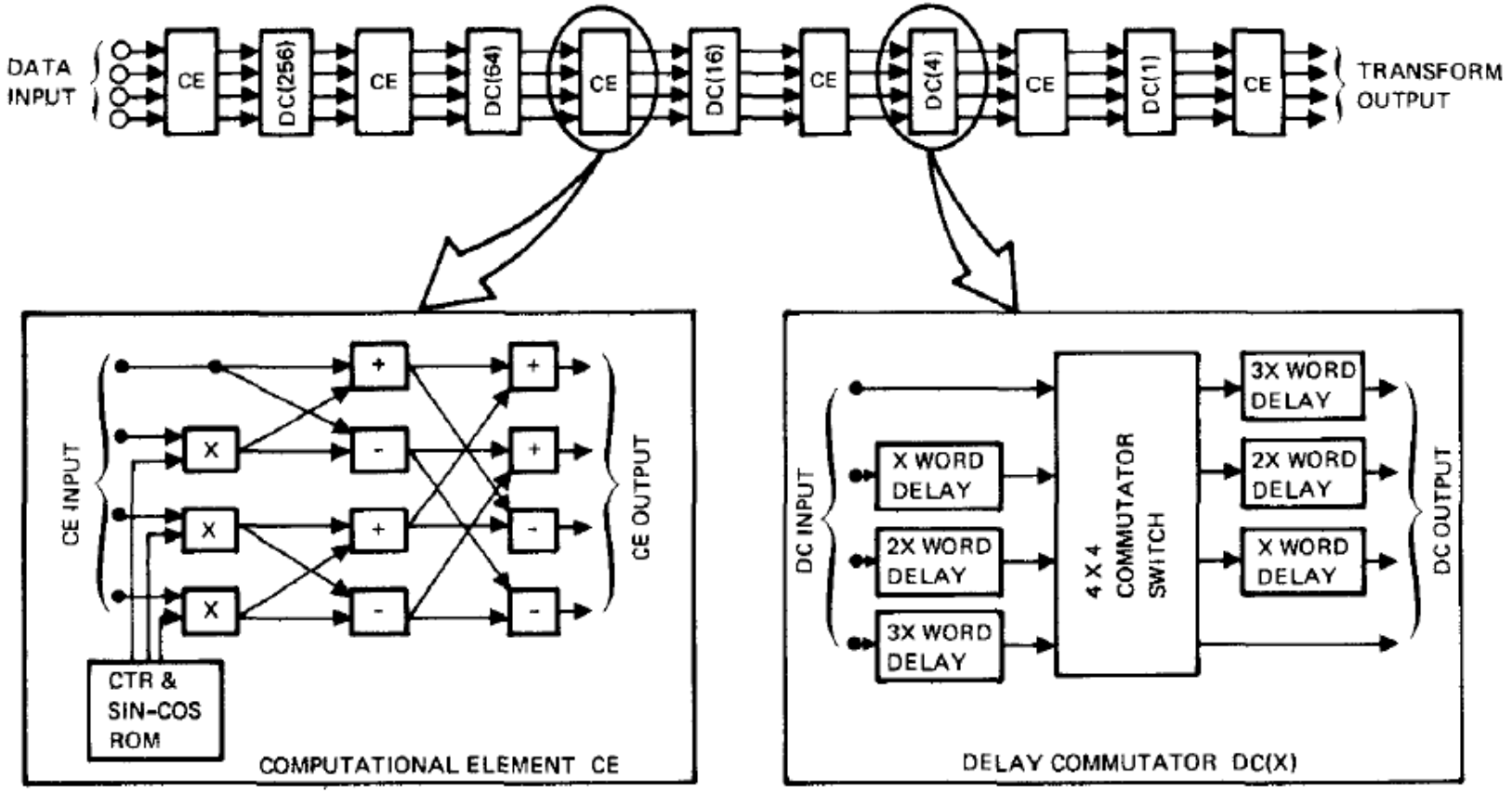


How does the Commutator Work? (2/2)



1024-Point Radix-4 FFT

Radix-4

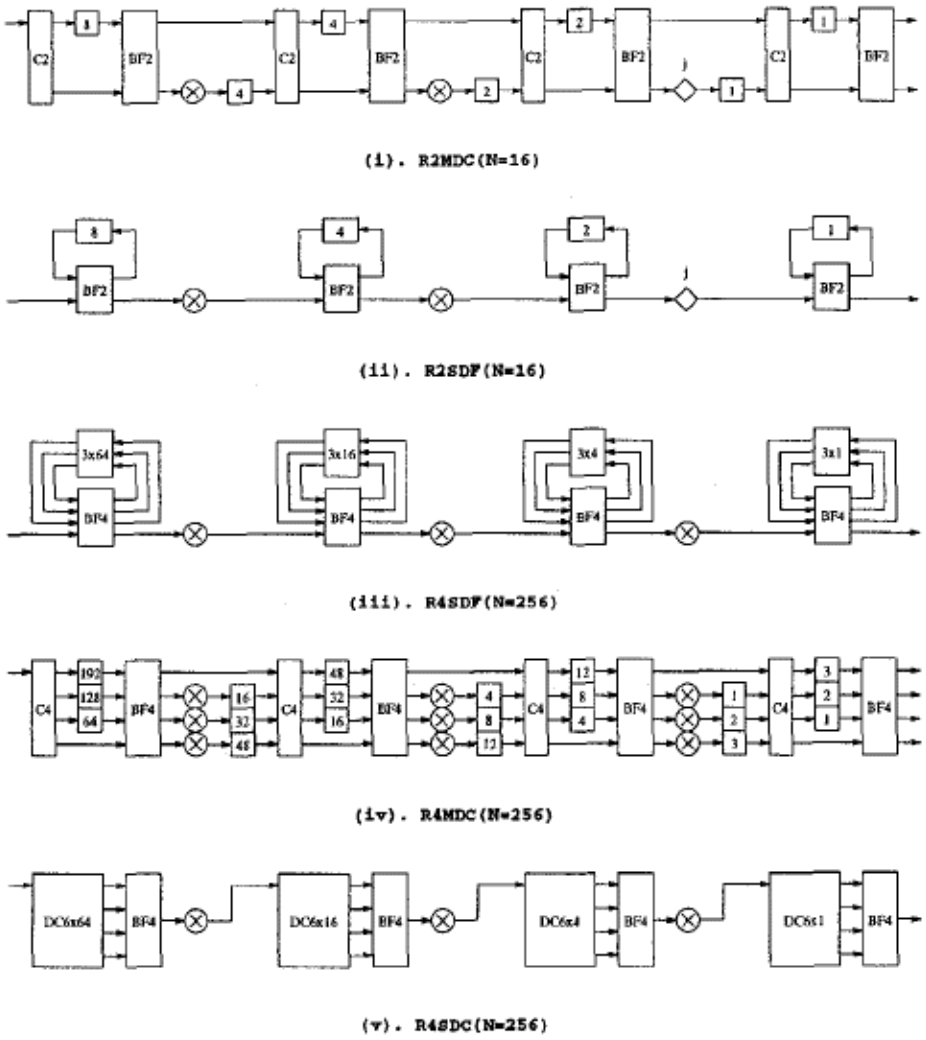


Drawbacks of Pipeline FFT

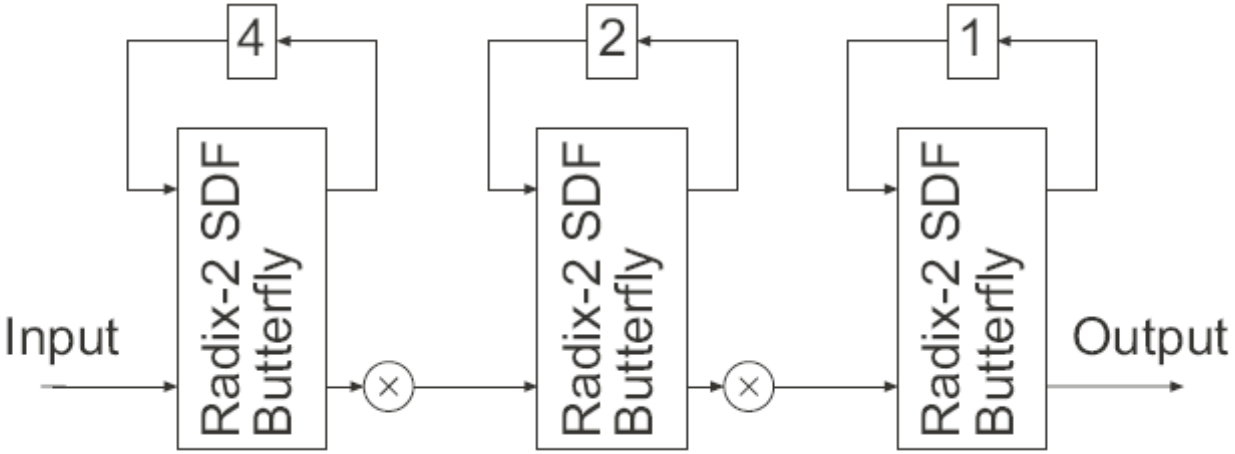
- Parallel input data is needed
- If the data is input serially, the hardware utilization will be low
  - For example: 25% for radix-4 FFT
- For the feedfoward architecture, large amount of memory is required for the data commutator
  - For example, 5/2N-4 register for radix-4 FFT

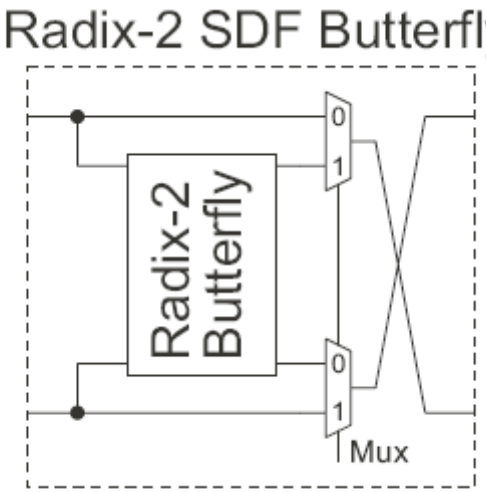
Alternative Forms

- **R2MDC**: radix-2 multi-path delay commutator
- **R2SDF**: radix-2 single path delay feedback
- **R4SDF**: radix-4 single-path delay feedback
- **R4MDC**: radix-4 multi-path delay commutator
- **R4SDC**: radix-4 single path delay commutator



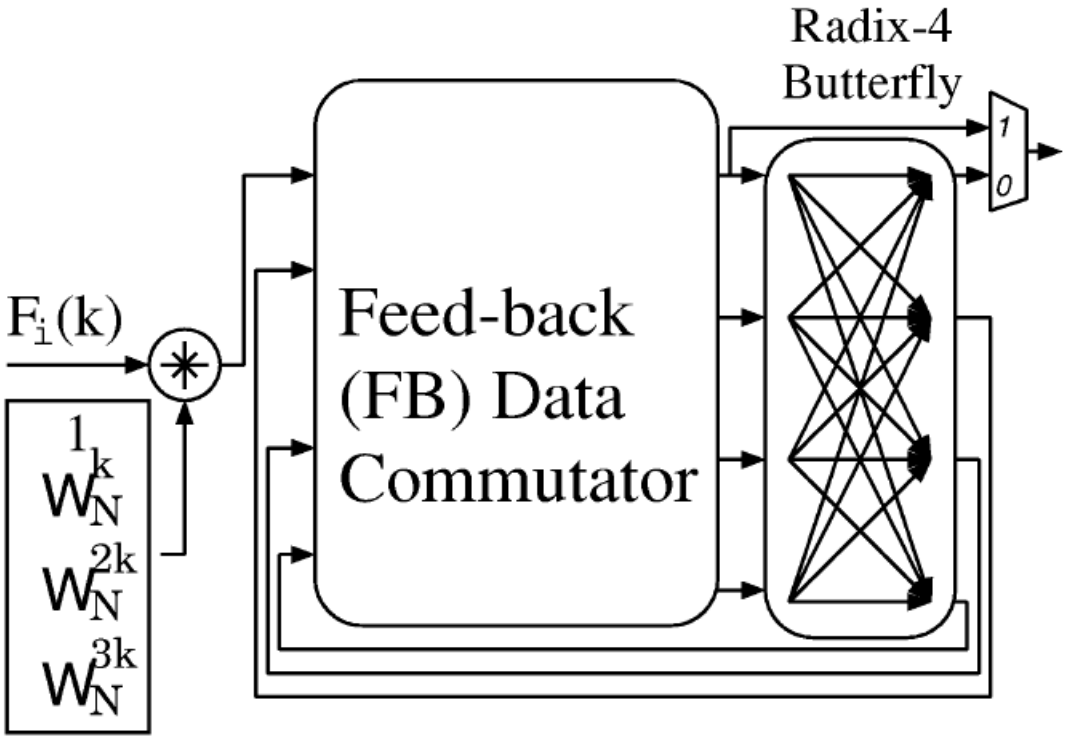
Feedback Architecture: R2SDF





Feedback Architecture: R4SDF

- Single path (word-serial)
- Can reduce the memory requirement by memory sharing



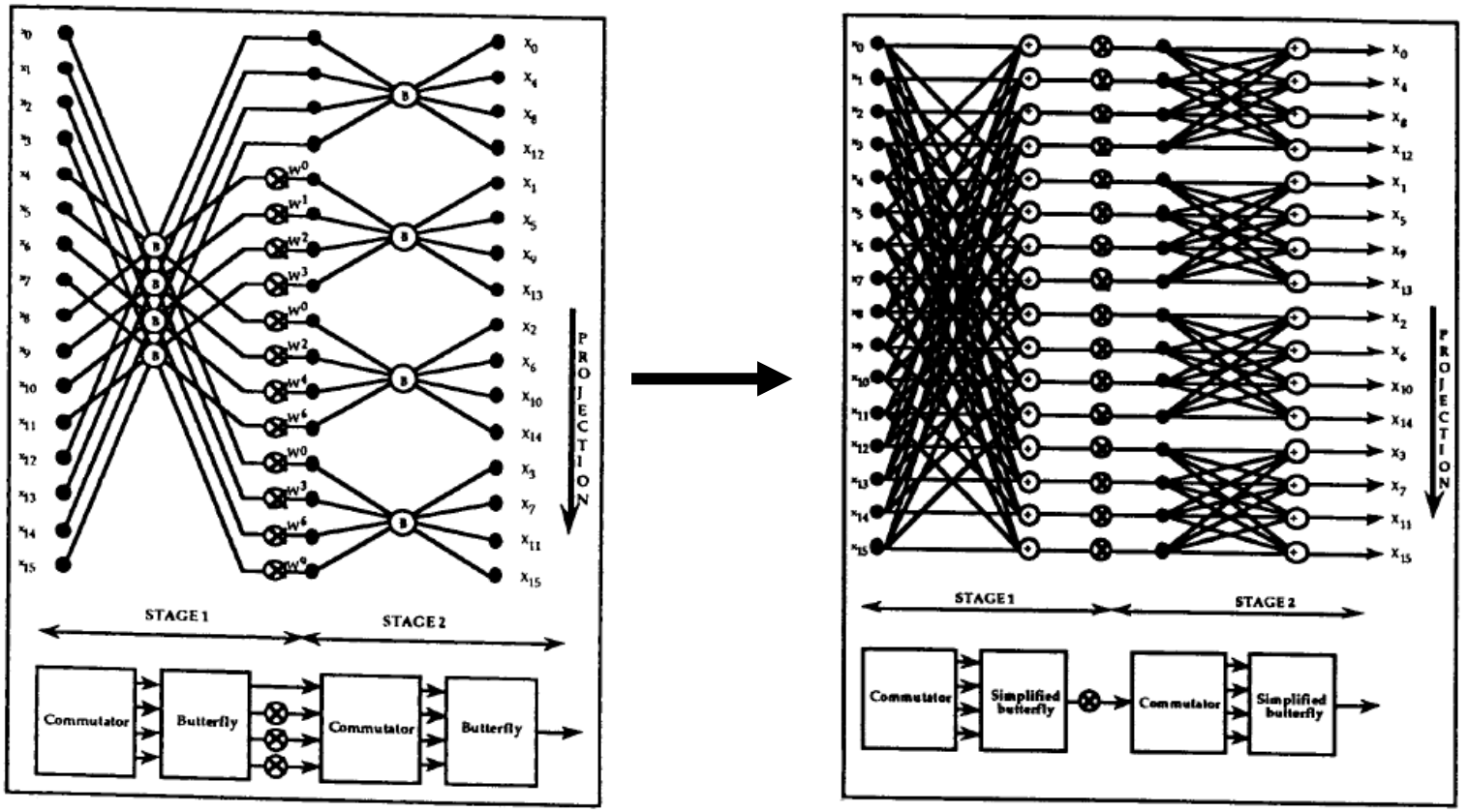
$$\begin{bmatrix} X[k] \\ X[k+N/4] \\ X[k+N/2] \\ X[k+3N/4] \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} W_N^0 F_0(k) \\ W_N^k F_1(k) \\ W_N^{2k} F_2(k) \\ W_N^{3k} F_3(k) \end{bmatrix}$$

Example 1

E. Bidet, D. Castelain, C. Joanblanq, and P. Seen, “A fast single-chip implementation of 8192 complex point FFT,” *IEEE Journal of Solid-State Circuits*, vol. 30, no. 3, March 1995.

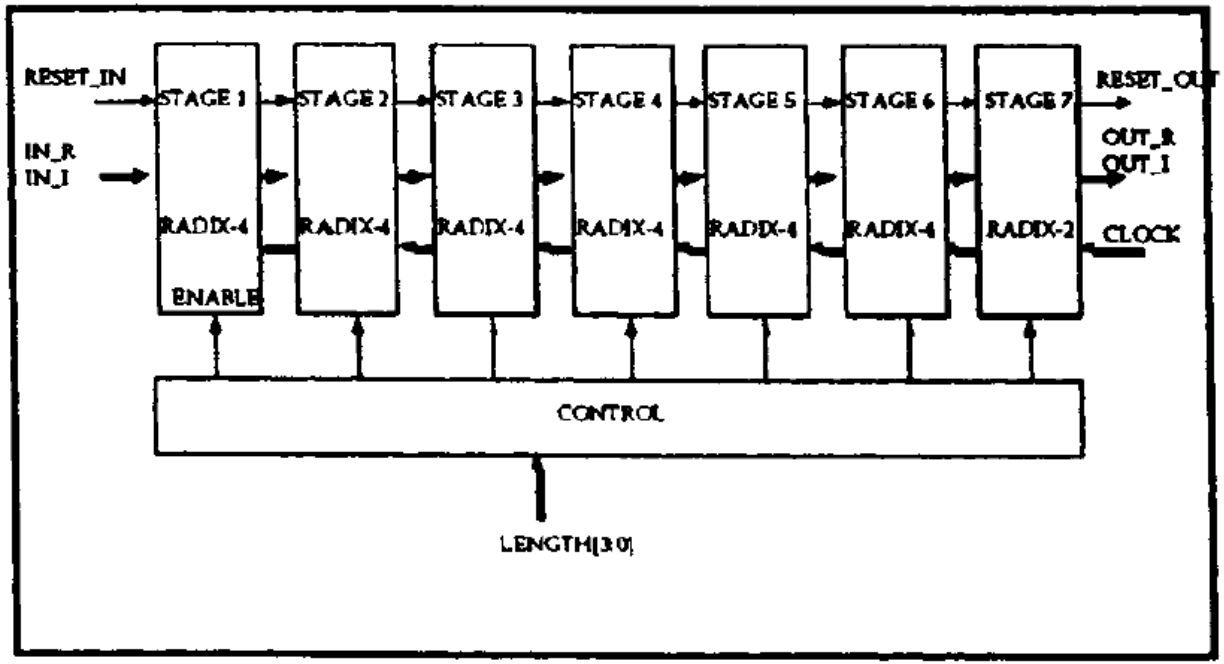
Radix-4 Single Path Delay Commutator

- Can increase the hardware utilization of the complex multipliers



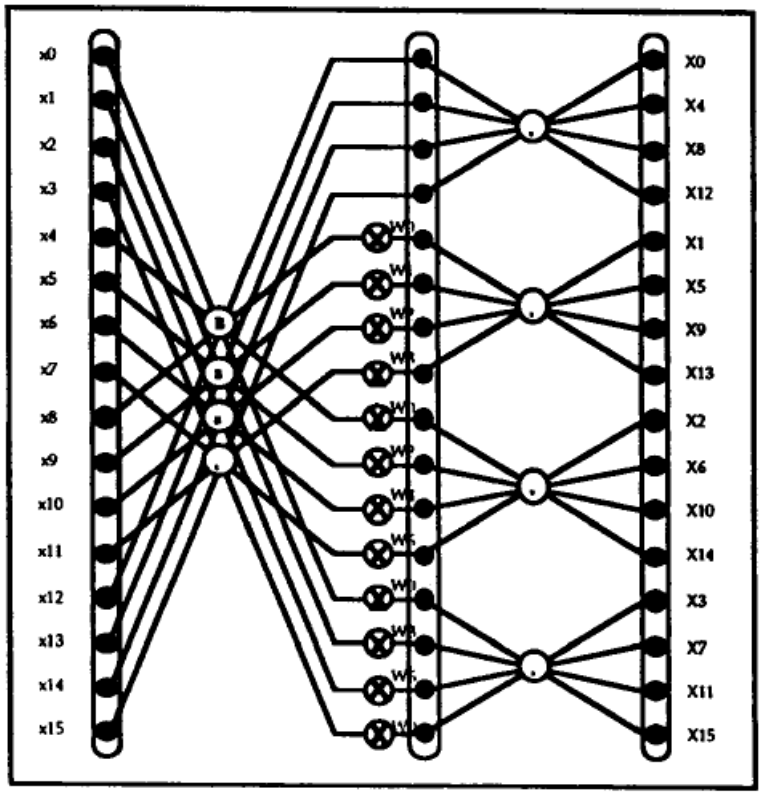
Chip Architecture

- $8192=4^6x2$



Block Floating Point (BFP)

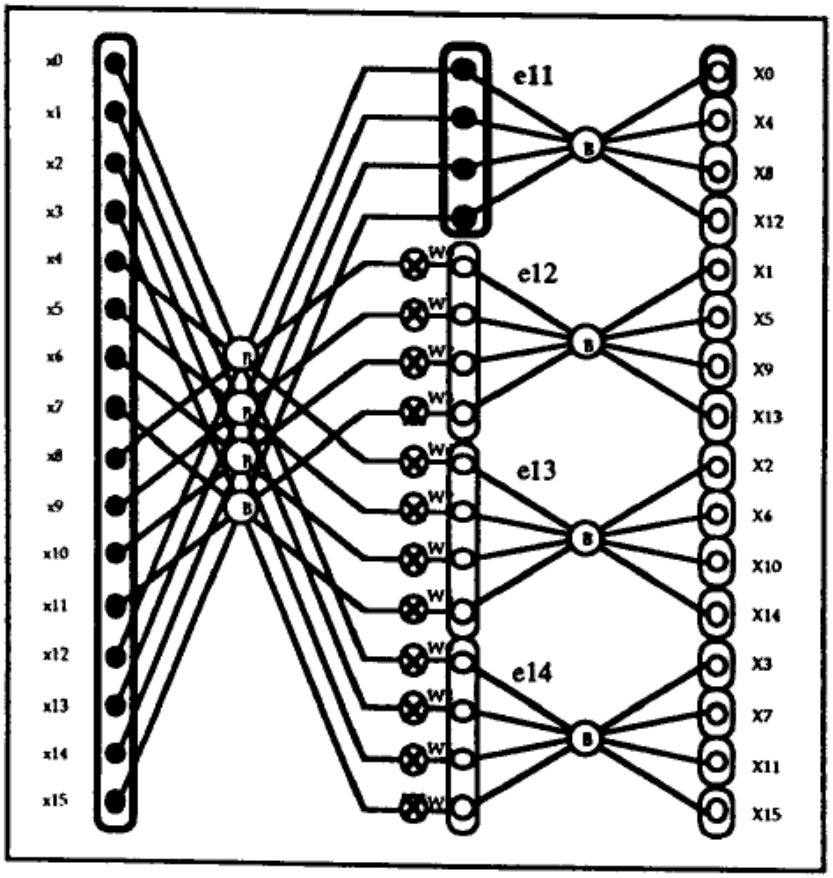
- Avoid to use floating point data format, use semi-floating point solution
  - Scaled fixed-point representation
  - Data have to be saturated and rounded after each stage
- Block floating point
  - Fixed point operation in each stage
  - Adaptively scaled depending on thee maximum amplitude of the data inside the block



There is a single exponent shared by all data in a block

Convergent Block Floating Point

- Attribute different rescaling factors to blocks of data during their transfer in the delay commutator



Summary

- Exploit radix-4 and radix-2 hybrid architecture for 8192-point FFT
- Apply Radix-4 Single Path Delay Commutator to reduce hardware cost and increase hardware utilization
- Use Convergent Block Floating Point to increase the precision of the calculation

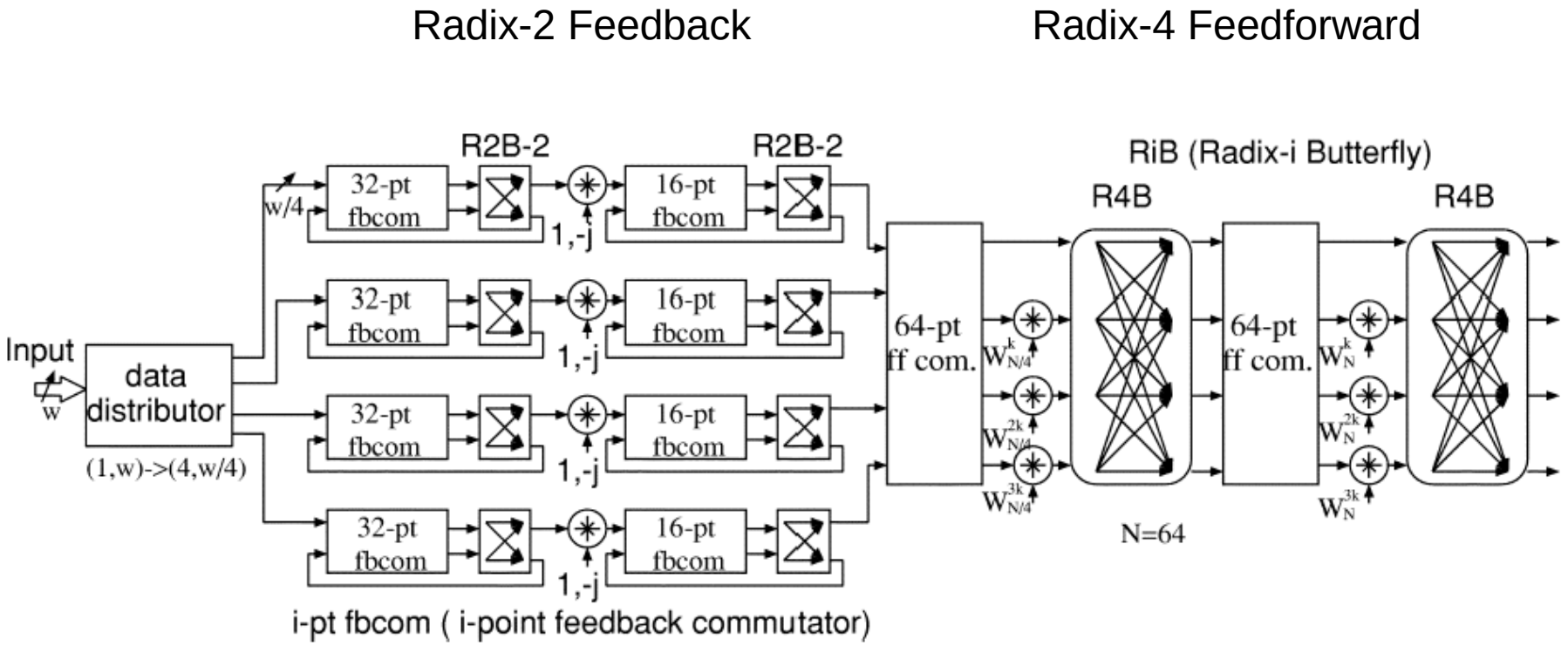
Example 2

Y.-N. Chang and K. K. Parhi, “An efficient pipelined FFT architecture,” *IEEE Transactions on Circuits and Systems---II: Analog and Digital Signal Processing*, vol. 50, no. 6, June 2003.

Key Concepts

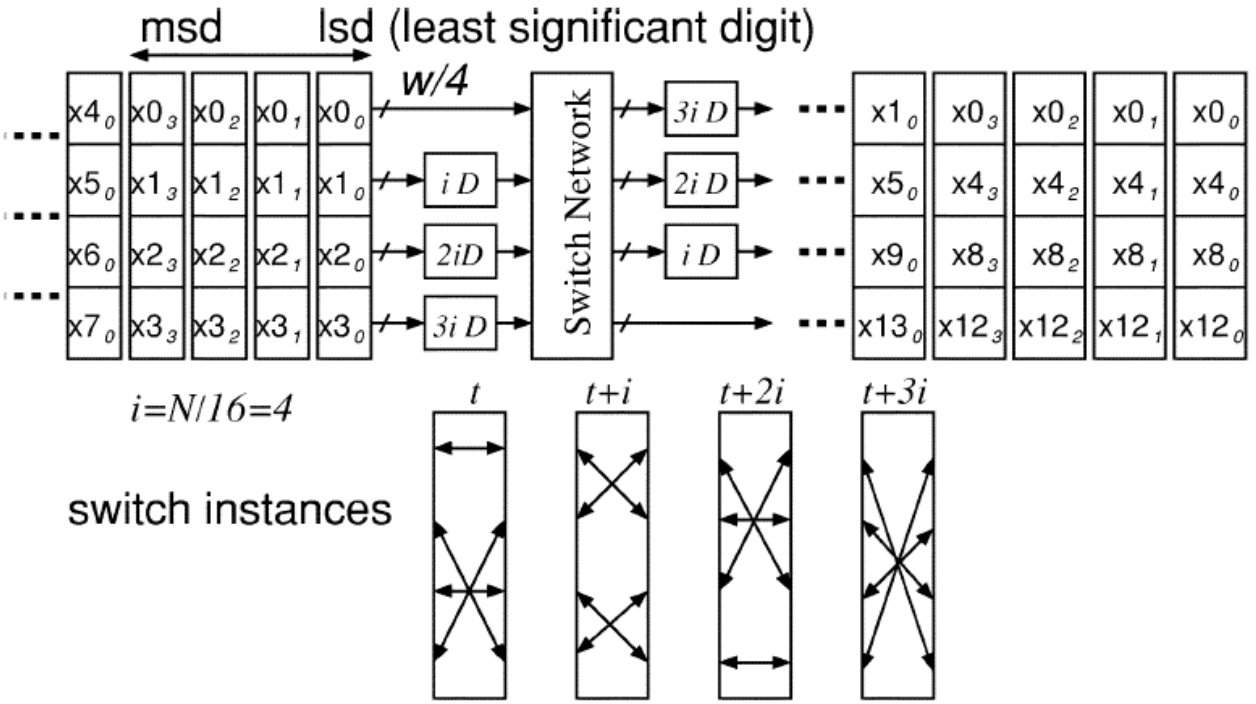
- To increase the hardware utilization, apply word-parallel/digit-serial arithmetic
  - 4 data in parallel, 4-bit for each data at each cycle
- Feedforward and feedback hybrid architecture
  - Feedforward: better hardware utilization
  - Feedback: less memory requirement

64-Point FFT Chip Architecture (1/3)



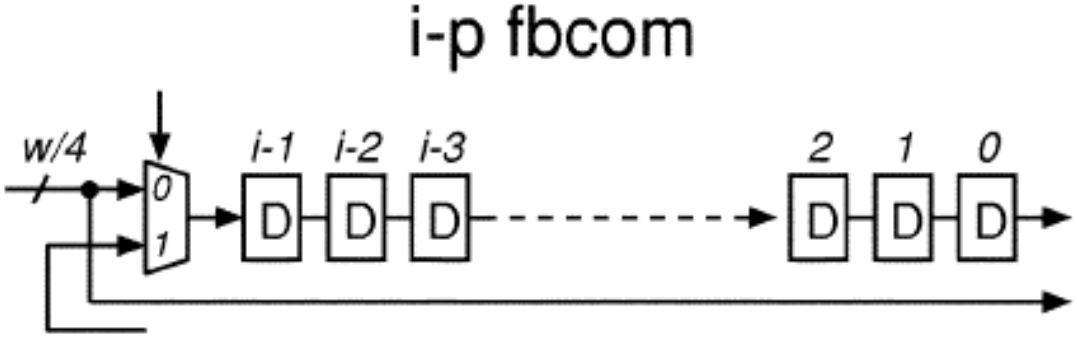
64-Point FFT Chip Architecture (2/3)

- Feedforward commutator

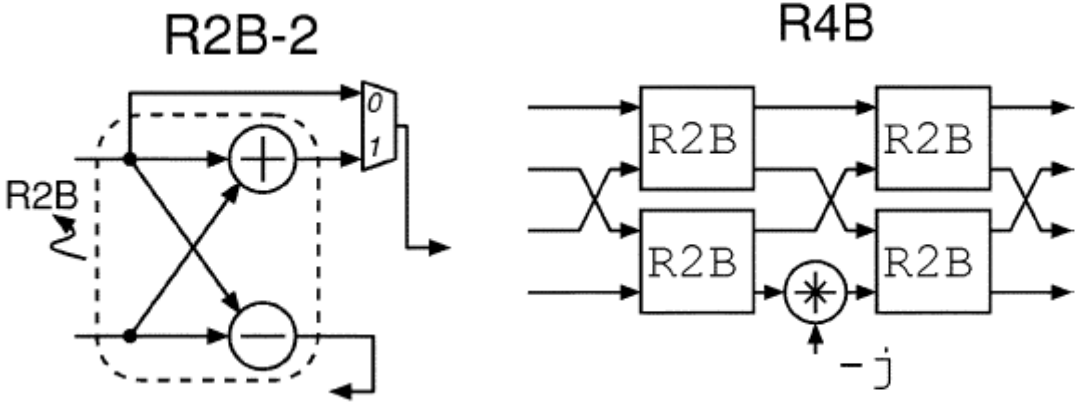


64-Point FFT Chip Architecture (3/3)

- Feedback commutator



- Butterflies



ROM Size Reduction

- The twiddle factors for W_N^{ik} and $W_N^{i((N/4)-k)}$ can share the same coefficient values

$$
\begin{array}{lll}
 & W_N^{ik} & W_N^{i((N/4)-k)} \\
i=1 & \cos\frac{2\pi k}{N}+j\sin\frac{2\pi k}{N} & \sin\frac{2\pi k}{N}+j\cos\frac{2\pi k}{N} \\
i=2 & \cos\frac{4\pi k}{N}+j\sin\frac{4\pi k}{N} & -\cos\frac{4\pi k}{N}+j\sin\frac{4\pi k}{N} \\
i=3 & \cos\frac{6\pi k}{N}+j\sin\frac{6\pi k}{N} & -\sin\frac{6\pi k}{N}+j\cos\frac{6\pi k}{N}.
\end{array}
$$

Comparison

TABLE I
COMPARISON OF PIPELINED FFT ARCHITECTURES (* BIT-PARALLEL, ** DIGIT-SERIAL)

	Bi & Jones [7]	He & Torkelson [5]	Hui et al [4]	Proposed
Commutator scheme	feedfoward	feedback	feedforward	mixed
Complex adders	$8log_4N^*$	$4log_4N^*$	$12log_4N^{**}$	$8(log_4N+1)^{**}$
Complex multipliers	log_4N-1^*	log_4N-1^*	$3(log_4N-1)^{**}$	$3(log_4N-1)^{**}$
Data Memory	$2.75N$	N	$2.5N$	$1.18N$
Twiddle factor ROM	N	N	N	$0.5N$
Computational efficiency	add 50% mul 75%	add 50% mul 75%	add 100% mul 100%	add~100% mul 100%

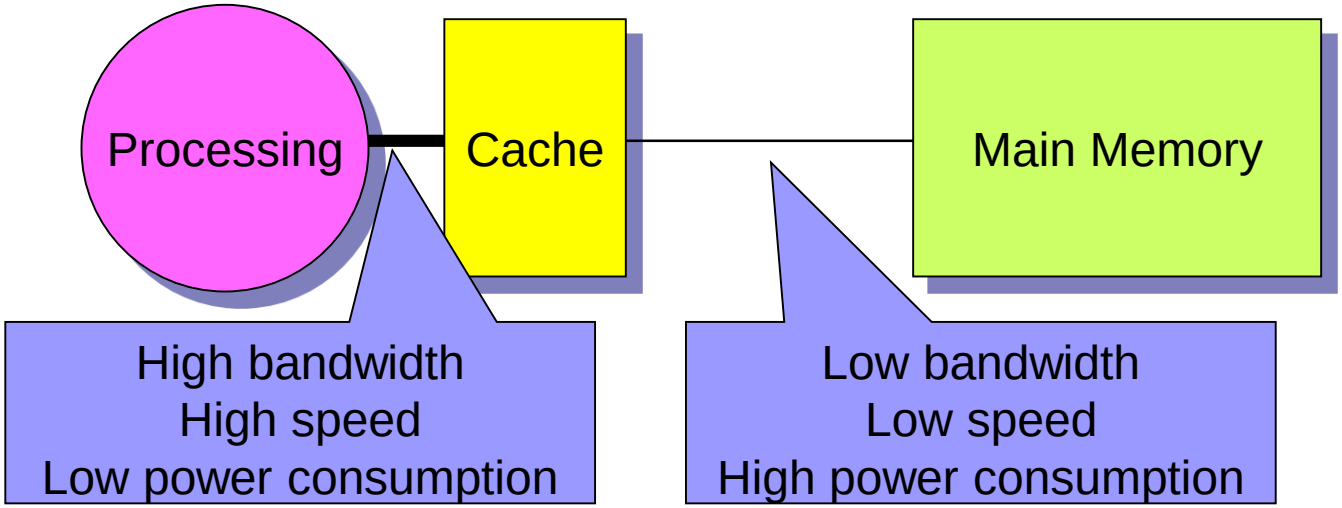
Summary

- Exploit digit-serial arithmetic to improve hardware utilization
- Feedforward/feedback mixed architecture
- ROM reduction technique

Cached-Memory Architecture

B. M. Bass, “A low-power, high performance, 1024-point FFT processor,” *IEEE Journal of Solid-State Circuits*, vol. 34, no. 3, March 1999.

Cached-Memory Architecture



- Advantages
  - Increased speed
  - Incresed energy efficiency
- Disadvantages
  - Addition of new functional units (caches)
  - Increase controller complexity

Cached-FFT Algorithm

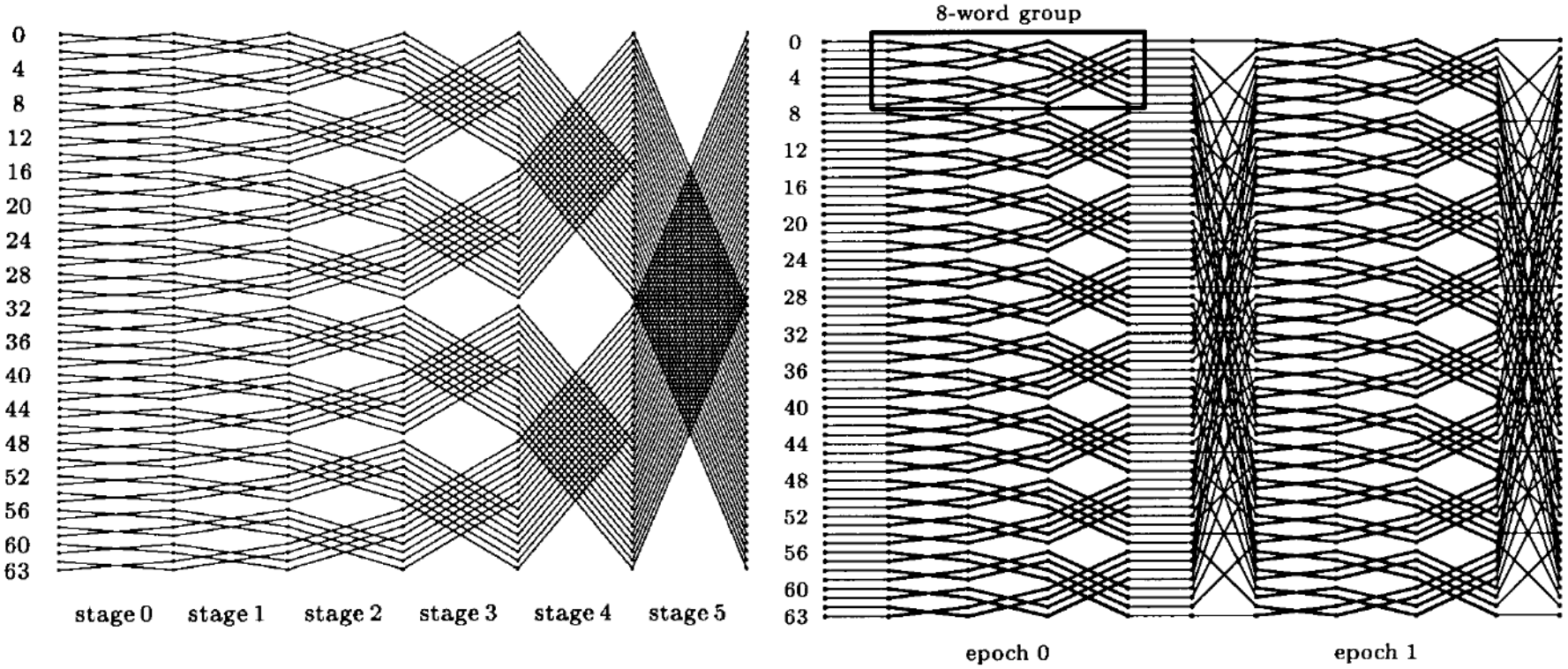
- To increase the efficiency of the cache, the memory access pattern should exhibit sufficient locality
- Nearly all FFT algorithms have poor locality
- Develop a new FFT algorithm first

Cached-FFT Algorithm

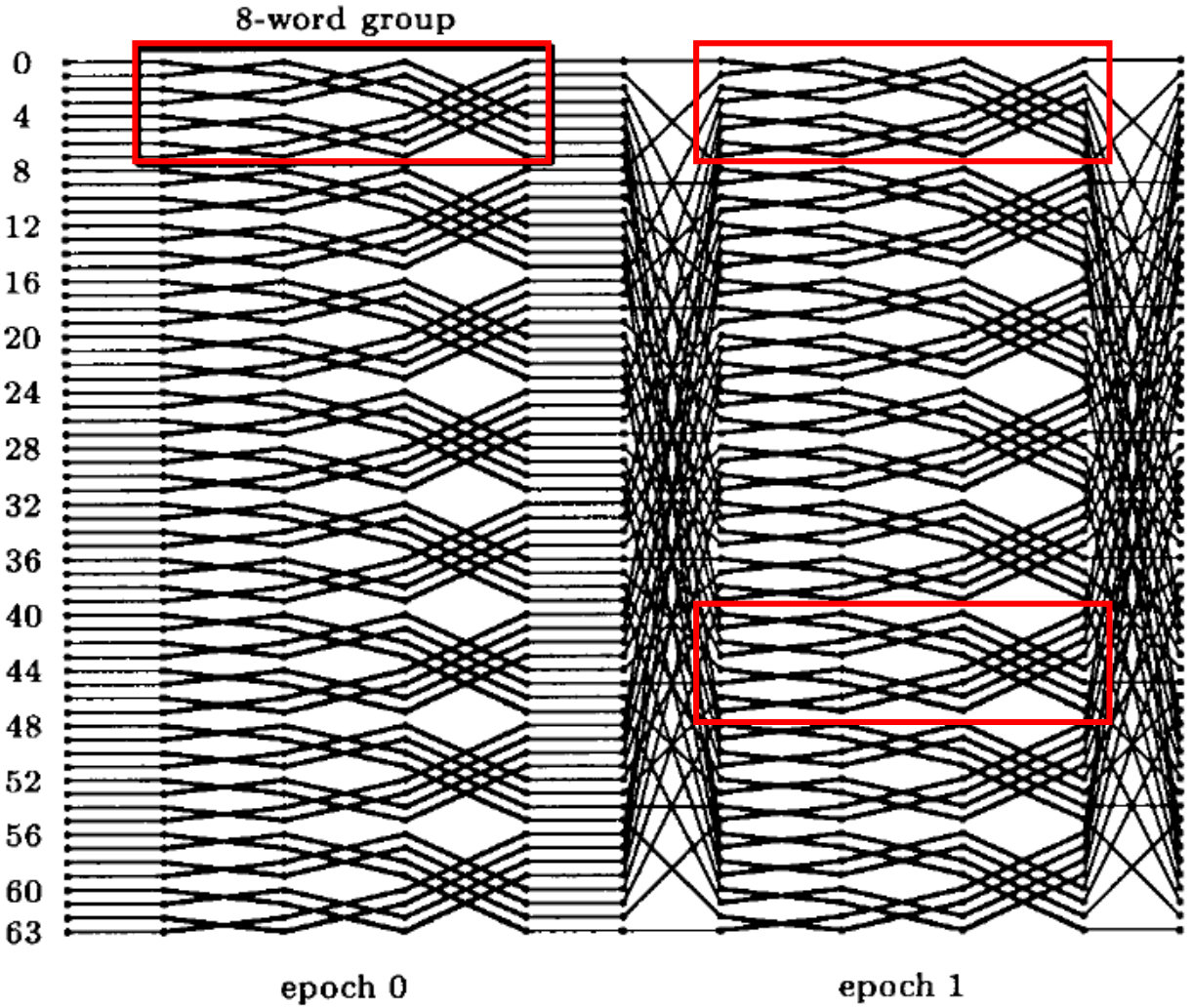
- Definition
  - Epoch, E, is a portion of FFT
  - A epoch should include several stages
  - All N data word are loaded into a cache, processed, and written back to main memory once
- Cache size $C = \sqrt[E]{N}$
- Memory traffic reduction multiple $(\log_r N)/E$

Cached-FFT Algorithm

- Example: 64-point FFT with E=2, $C = \sqrt[2]{64} = 8$

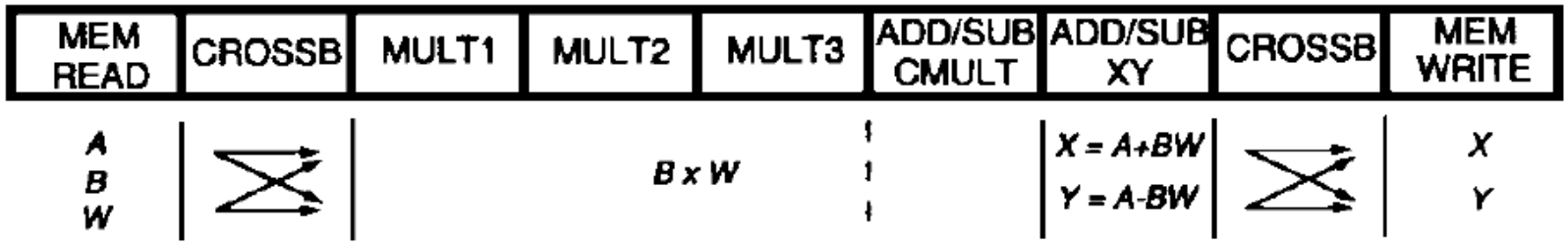


Cached-FFT Algorithm



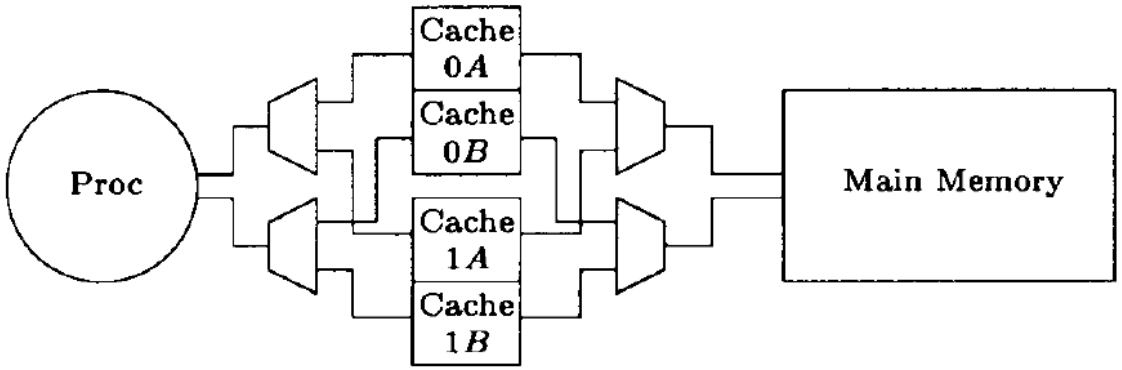
Processor Architecture

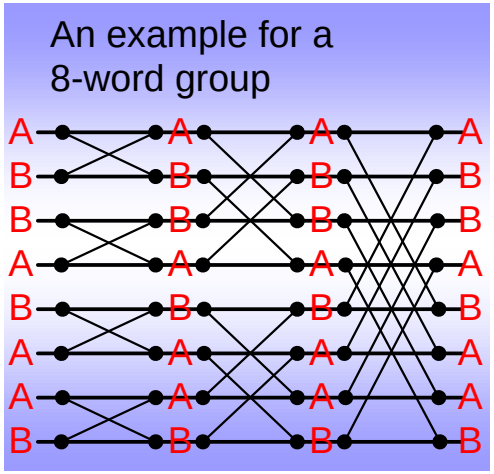
- 9-stage pipeline
- The datapath calculate one complex radix-2 DIT butterfly per cycle
- Fixed-point datapth has widths varying from 20 to 24 bits



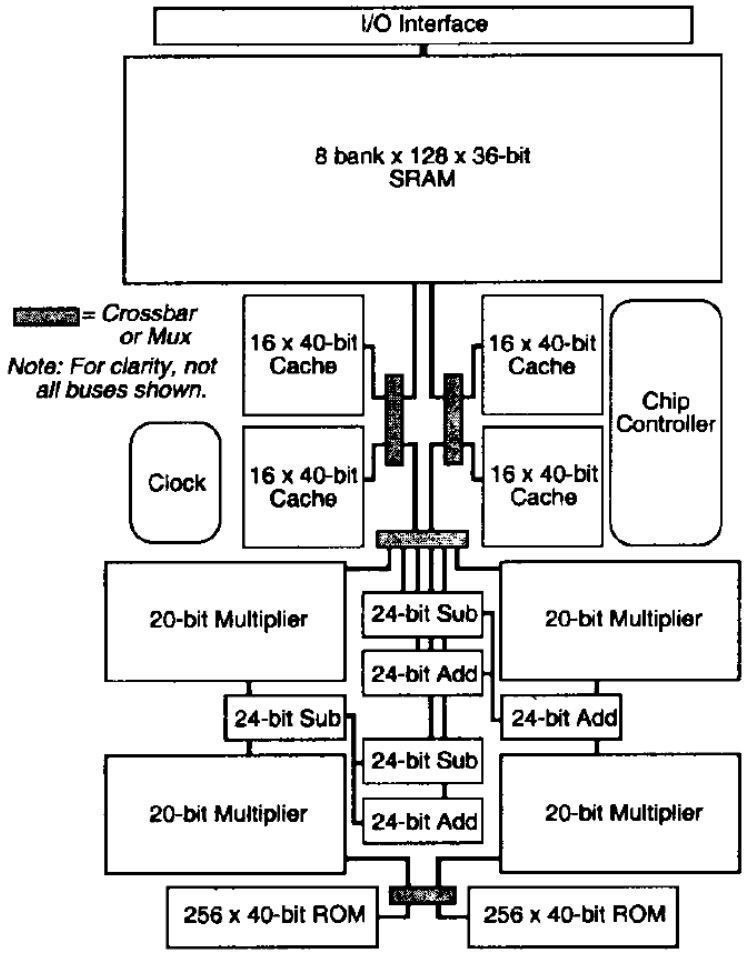
Memory System

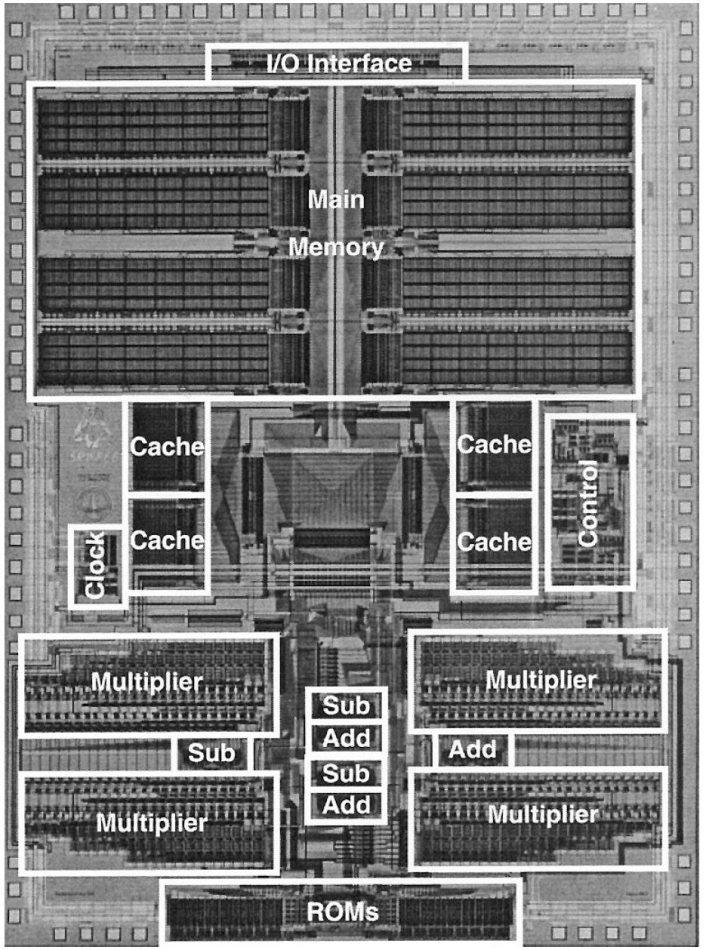
- Ping-pong mode configuration can increase processor utilization
- Double-banked arrangement can increase the throughput





Chip Block Diagram and Die Photo



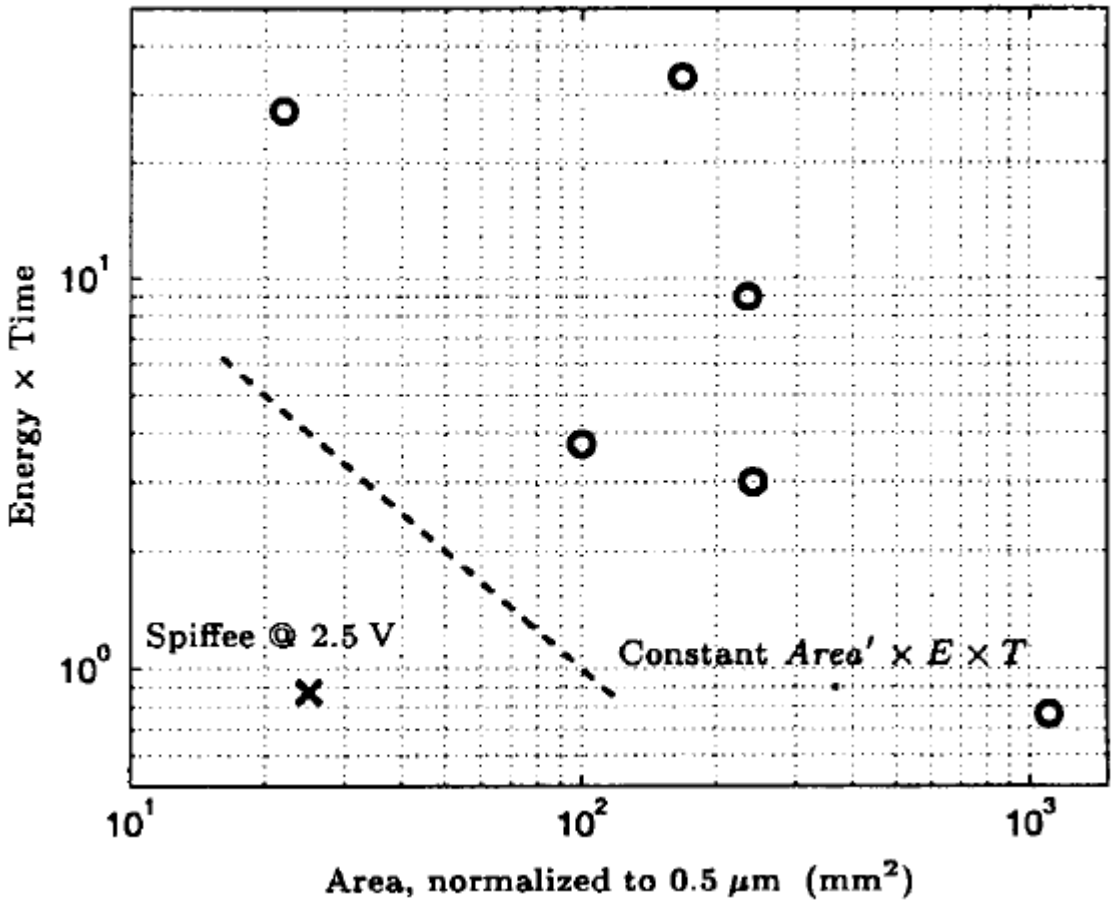


Comparison (1)

TABLE II
COMPARISON OF PROCESSORS CALCULATING 1024-POINT COMPLEX FFT's

Processor	CMOS Tech (μm)	Datapath width (bits)	1024-pt Exec Time (μsec)	Power (mW)	Clock Freq (MHz)	Num of chips	Norm Area (mm^2)	FFTs per Energy
LSI, L64280 [5]	1.5	20	26	20,000	40	20	233	2.9
Plessey, 16510A [9]	1.4	16	98	3,000	40	1	22	3.6
Honeywell, DASP [3]	1.2	16	102	$\sim$ 5,250	–	2+	–	1.7
Y. Zhu, U of Calgary	1.2	16	155	–	33	–	–	–
Dassault Electronique	1.0	12	10.2	15,000	25	6	240	3.4
Tex Mem Sys, TM-66	0.8	32	65	7,000	50	2+	–	3.4
Cobra, Col. State [8]	0.75	23	9.5	7,700	40	16+	1104+	12.4
Sicom, SNC960A	0.6	16	20	2,500	65	1	–	9.0
CNET, E. Bidet [7]	0.5	10	51	300	20	1	100	13.6
M. Wosnitza, ETH [23]	0.5	32	80	6000	66	1	167	2.4
Spiffee, $V_{dd} = 3.3V$	0.7/0.6	20	30	845	173	1	25	27.6
Spiffee, $V_{dd} = 2.5V$	0.7/0.6	20	41	363	128	1	25	47.0
Spiffee, $V_{dd} = 1.1V$	0.7/0.6	20	330	9.5	16	1	25	**223**

Comparison (2)



Summary

- Multi-level memory hierarchy is to improve the memory access efficiency
- Cached-FFT is designed to improve the data locality
- Memory system
  - Ping-pong mode can improve utilization
  - Double bank can increase the throughput
- Deep pipelined processor