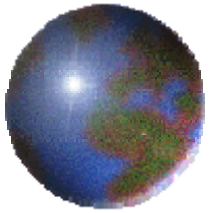


Introduction to FFT Processors

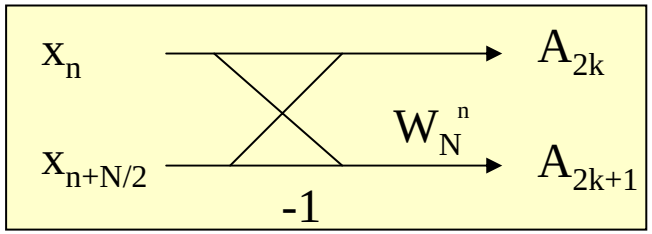
Chih-Wei Liu
VLSI Signal Processing Lab
Department of Electronics Engineering
National Chiao-Tung University

FFT Design

- FFT
  - Consists of a series of complex additions and complex multiplications
- Algorithm
  - Cooley-Tukey decomposition for power of two length FFT
- Architecture
  - Systematic mapping procedure

Algorithm Level

- Cooley-Tukey decomposition
  - Radix-2, decimation-in-frequency

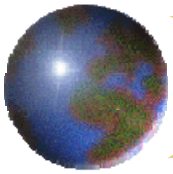


$$A_{2k} = \sum_{n=0}^{N-1} x_n W_N^{n2k} = \sum_{n=0}^{N/2-1} (x_n + x_{n+N/2}) W_{N/2}^{nk}$$

$$A_{2k+1} = \sum_{n=0}^{N-1} x_n W_N^{n(2k+1)} = \sum_{n=0}^{N/2-1} (x_n - x_{n+N/2}) W_N^n W_{N/2}^{nk}$$

- Variants based on CT algorithm
  - **Fixed radix**: Radix-2, Radix-4, Radix-8, Radix-2^2
  - **Mixed radix**: Split-radix, Radix-2/8, Radix-2/4/8
  - Number of addition
    - Same for any mixed-radix or fixed-radix algorithm.
  - Number of multiplication
    - Depends on the reduction of trivial multiplications.

Hence, increase additions



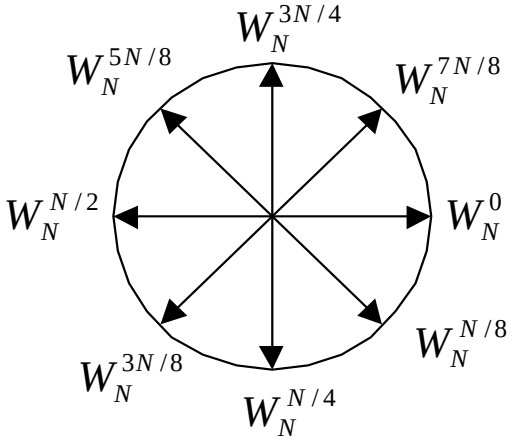
FFT Algorithms

- Review of Radix-2^r algorithm
  - DIF(decimation in frequency) and DIT(decimation in time) version
  - Radix-2 algorithm
  - Radix-4 and Radix-2^2 algorithm
  - Radix-8 and Radix-2^3 algorithm
  - Split-radix 2/4 and Split-radix 2/8

FFT Algorithms

- DFT

$$X(k)=\sum_{n=0}^{N-1}x(n)e^{-j\frac{2\pi}{N}kn}\equiv\sum_{n=0}^{N-1}x(n)W_N^{kn}\quad ,k=0,1,..N-1.$$



$$W_N^0=-W_N^{N/2}=1$$

$$W_N^{N/4}=-W_N^{3N/4}=-j$$

$$W_N^{N/8}=-W_N^{5N/8}=\frac{\sqrt{2}}{2}(1-j)$$

$$W_N^{3N/8}=-W_N^{7N/8}=-\frac{\sqrt{2}}{2}(1+j)$$

$$(a+jb)*W_8^1=\frac{\sqrt{2}}{2}[(a+b)+j(b-a)]$$

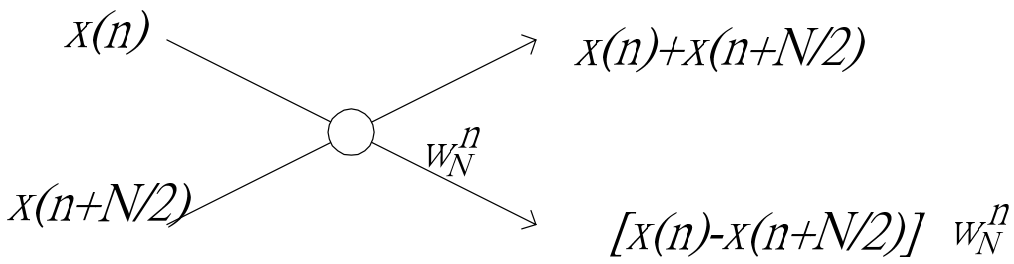
$$(a+jb)*W_8^3=\frac{\sqrt{2}}{2}[(b-a)-j(b+a)]$$

FFT Algorithms

- Radix-2 Algorithm
  - DIF Radix-2 Algorithm

$$\begin{cases} X(2k_l) = \sum_{n=0}^{N/2-1}[x(n)+x(n+N/2)]W_{N/2}^{k_l n} \\ X(2k_l+1) = \sum_{n=0}^{N/2-1}[x(n)-x(n+N/2)]W_N^n W_{N/2}^{k_l n} \end{cases} \qquad k_l = 0,1,\ldots,N/2-1.$$

- Butterfly of Radix-2 Algorithm
  - DIF Form



FFT Algorithms

- Radix-4 Algorithm

$$X(4k_1+l)=\sum_{k=0}^{N/4-1}[x(n)+x(n+\frac{N}{4})\times W_4^l+x(n+\frac{N}{2})W_4^{2l}+x(n+\frac{3N}{4})\times W_4^{3l}]W_N^{nl}W_{N/4}^{nk_1}$$

$$l=0,1,2,3;\ k_1=0\sim N/4-1;$$

- Radix-2^2 Algorithm

$$X(4k_1+2l_2+l_1)$$

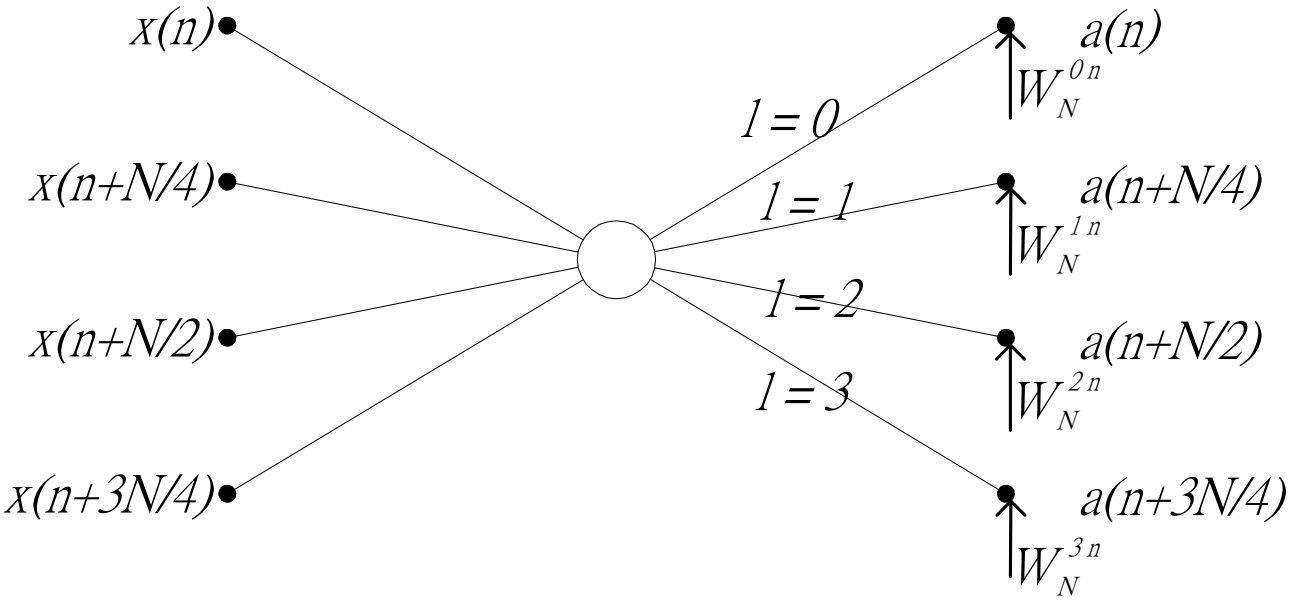
$$=\sum_{k=0}^{N/4-1}[x(n)+x(n+N/4)\times W_4^{2l_2+l_1}+x(n+N/2)W_4^{4l_2+2l_1}+x(n+3N/4)\times W_4^{6l_2+3l_1}]W_N^{n(2l_2+l_1)}W_{N/4}^{nk_1}$$

$$=\sum_{n=0}^{N/4-1}\{[x(n)+(-1)^{l_1}x(n+N/2)]+(-1)^{l_2}(-j)^{l_1}[x(n+N/4)+(-1)^{l_1}x(n+3N/4)]\}W_N^{n(2l_2+l_1)}W_{N/4}^{nk_1}$$

$$l_1,l_2=0,1;\ k_1=0\sim N/4-1.$$

FFT Algorithms

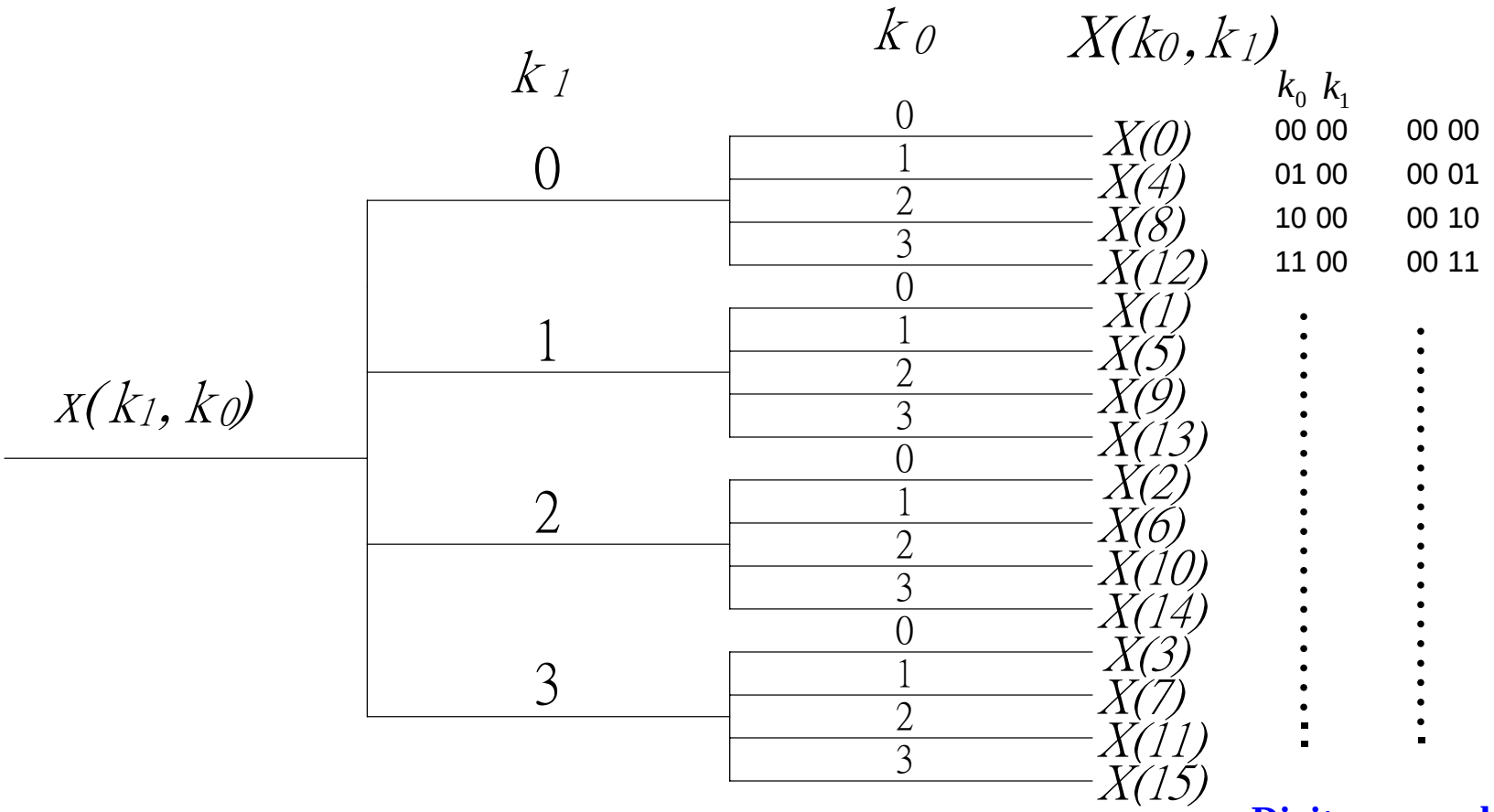
- Butterfly of Radix-4 Algorithm



(Data Ordering: Digit Reversed)

FFT Algorithms

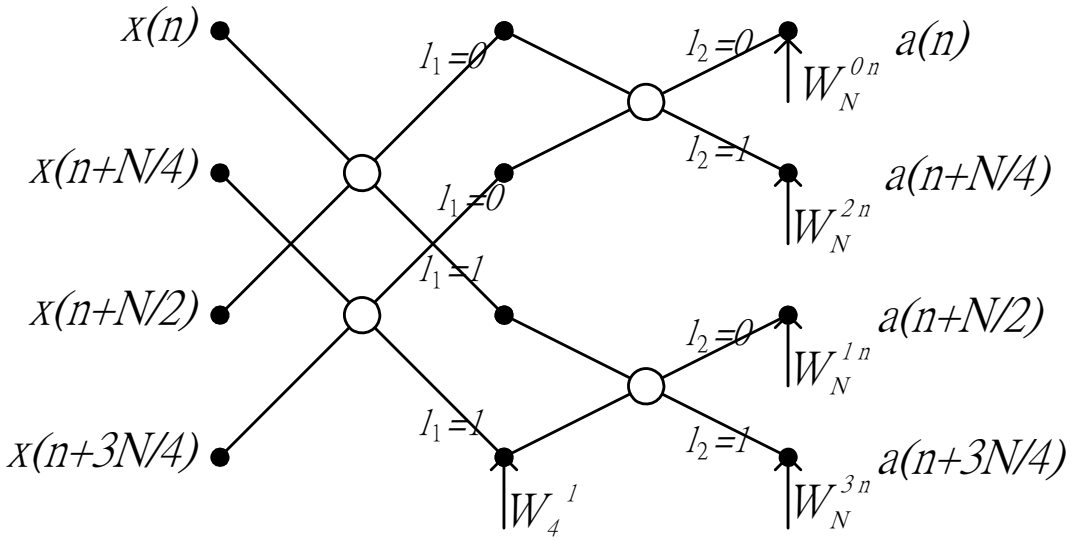
- Data Ordering of Radix-4 (N=16)



Digit-reversed ordering

FFT Algorithms

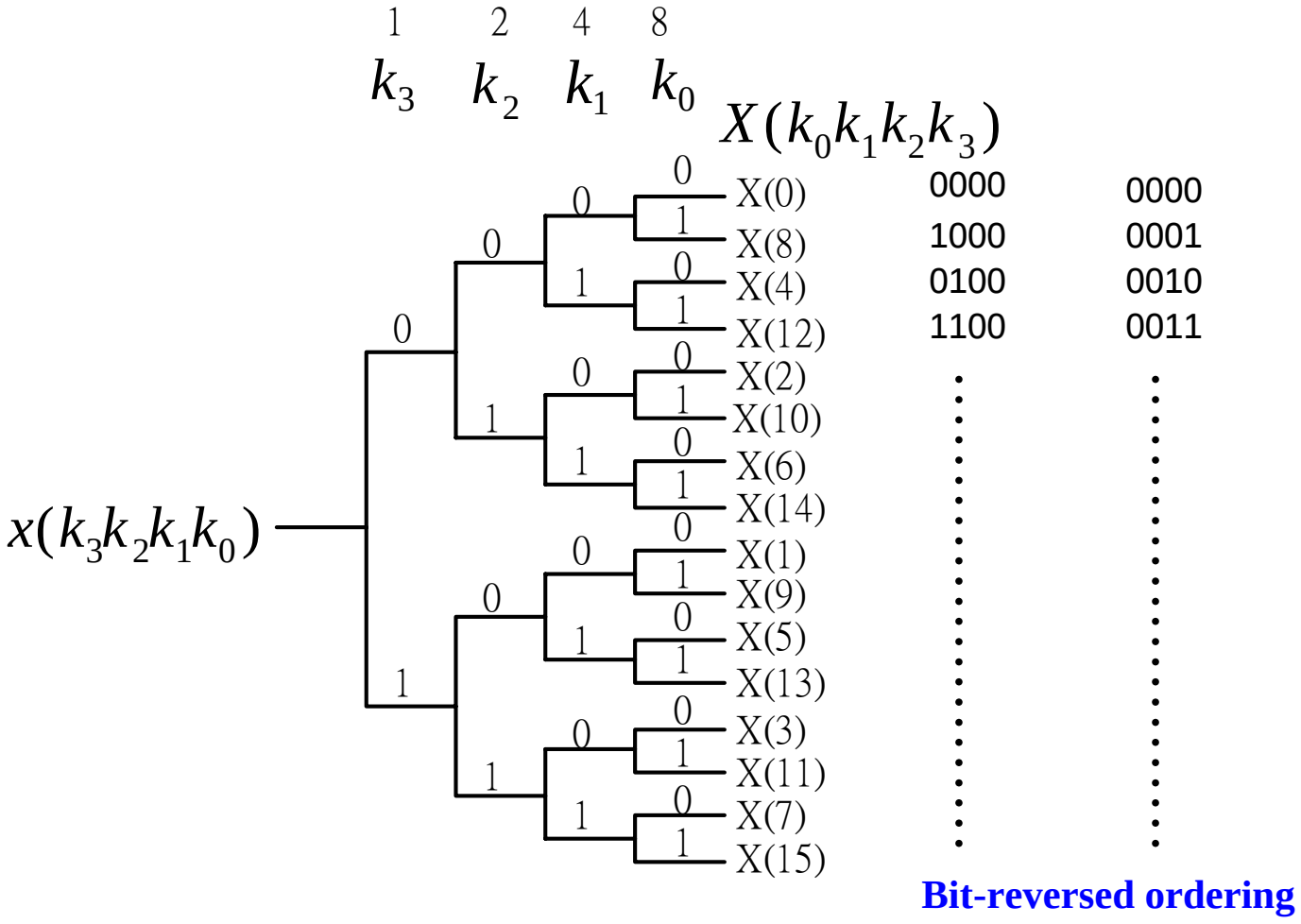
- Butterfly of radix-2^2 Algorithm



(Data Ordering: Bit Reversed)

FFT Algorithms

- Data Ordering of Radix- 2^2 (N=16)



FFT Algorithms

- DIF Radix-8 Algorithm

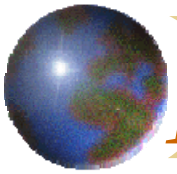
$$X(8k+l)=\sum_{n=0}^{N-1}x(n)W_N^{(8k+l)n}=\sum_{m=0}^{7}\sum_{n=0}^{N/8-1}x(n+\frac{mN}{8})W_N^{(8k+l)(mN/8+n)}$$

$$=\sum_{m=0}^{7}\sum_{n=0}^{N/8-1}[x(n+\frac{mN}{8})W_8^{lm}]W_N^{nl}W_{N/8}^{nk}$$

$$=\sum_{n=0}^{N/8-1}\{[x(n)+x(n+\frac{2N}{8})W_4^{l}+x(n+\frac{4N}{8})W_4^{2l}+x(n+\frac{6N}{8})W_4^{-l}]$$

$$+[x(n+\frac{N}{8})+x(n+\frac{3N}{8})W_4^{l}+x(n+\frac{5N}{8})W_4^{2l}+x(n+\frac{7N}{8})W_4^{-l}]W_8^{l}\}W_N^{nl}W_{N/8}^{nk}$$

$$l=0,1,2,3,4,5,6,7;\ k=0\sim N/8-1.$$



FFT Algorithms

- DIF Radix-2^3 Algorithm

$$X(8k+4l_3+2l_2+l_1)$$

$$= \sum_{n=0}^{N/8-1} \{[x(n)+x(n+\frac{2N}{8})W_4^l + x(n+\frac{4N}{8})W_4^{2l} + x(n+\frac{6N}{8})W_4^{-l}]$$

$$+[x(n+\frac{N}{8})+x(n+\frac{3N}{8})W_4^l + x(n+\frac{5N}{8})W_4^{2l} + x(n+\frac{7N}{8})W_4^{-l}]W_8^l\}W_N^{nl}W_{N/8}^{nk}$$

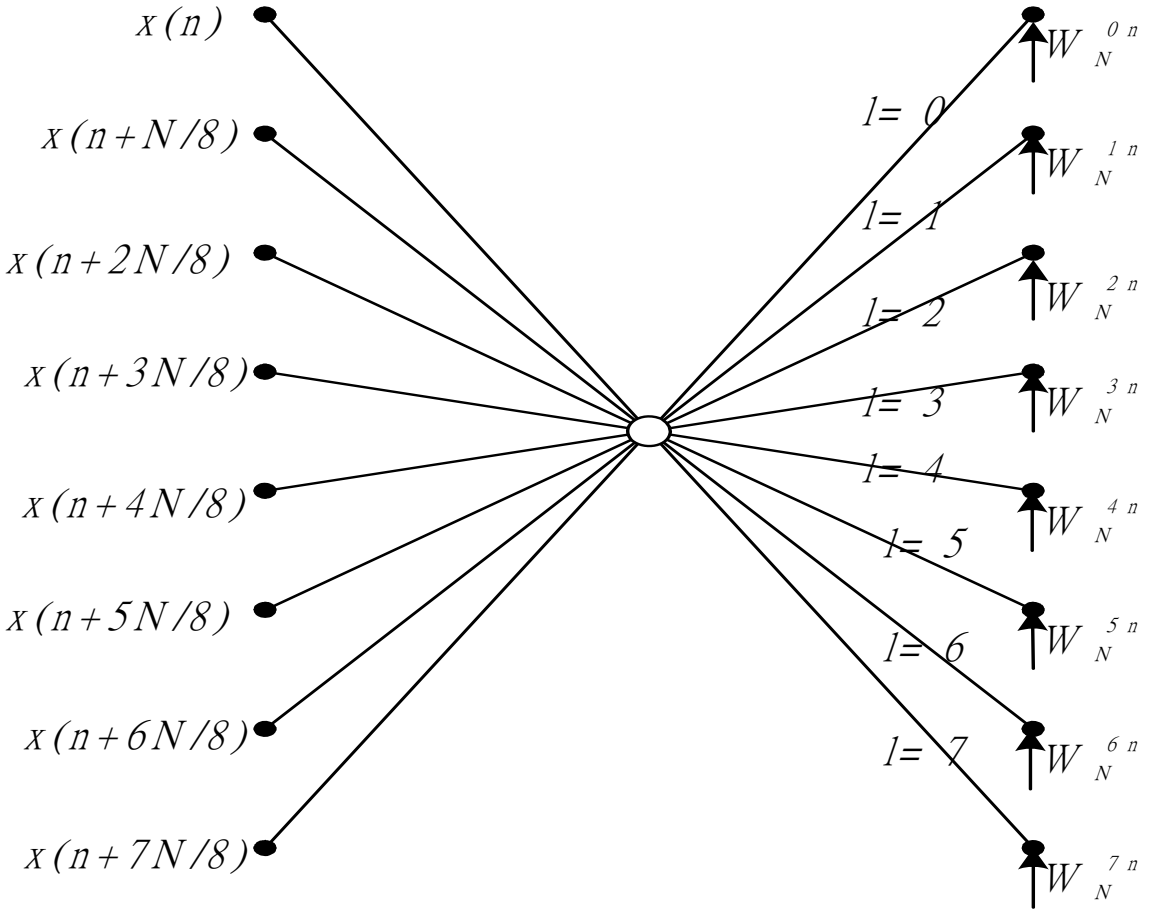
$$= \sum_{n=0}^{N/8-1} \{[(x(n)+W_2^{l_1}x(n+\frac{4N}{8}))+W_2^{l_2}W_4^{l_1}(x(n+\frac{2N}{8})+W_2^{l_1}x(n+\frac{6N}{8}))]$$

$$+[(x(n+\frac{N}{8})+W_2^{l_1}x(n+\frac{5N}{8}))+W_2^{l_2}W_4^{l_1}(x(n+\frac{3N}{8})+W_2^{l_1}x(n+\frac{7N}{8}))]W_8^{2l_2+l_1}\}W_N^{n(4l_3+2l_2+l_1)}W_{N/8}^{nk}$$

$$l_1, l_2, l_3 = 0,1; \quad k = 0 \sim N/8-1.$$

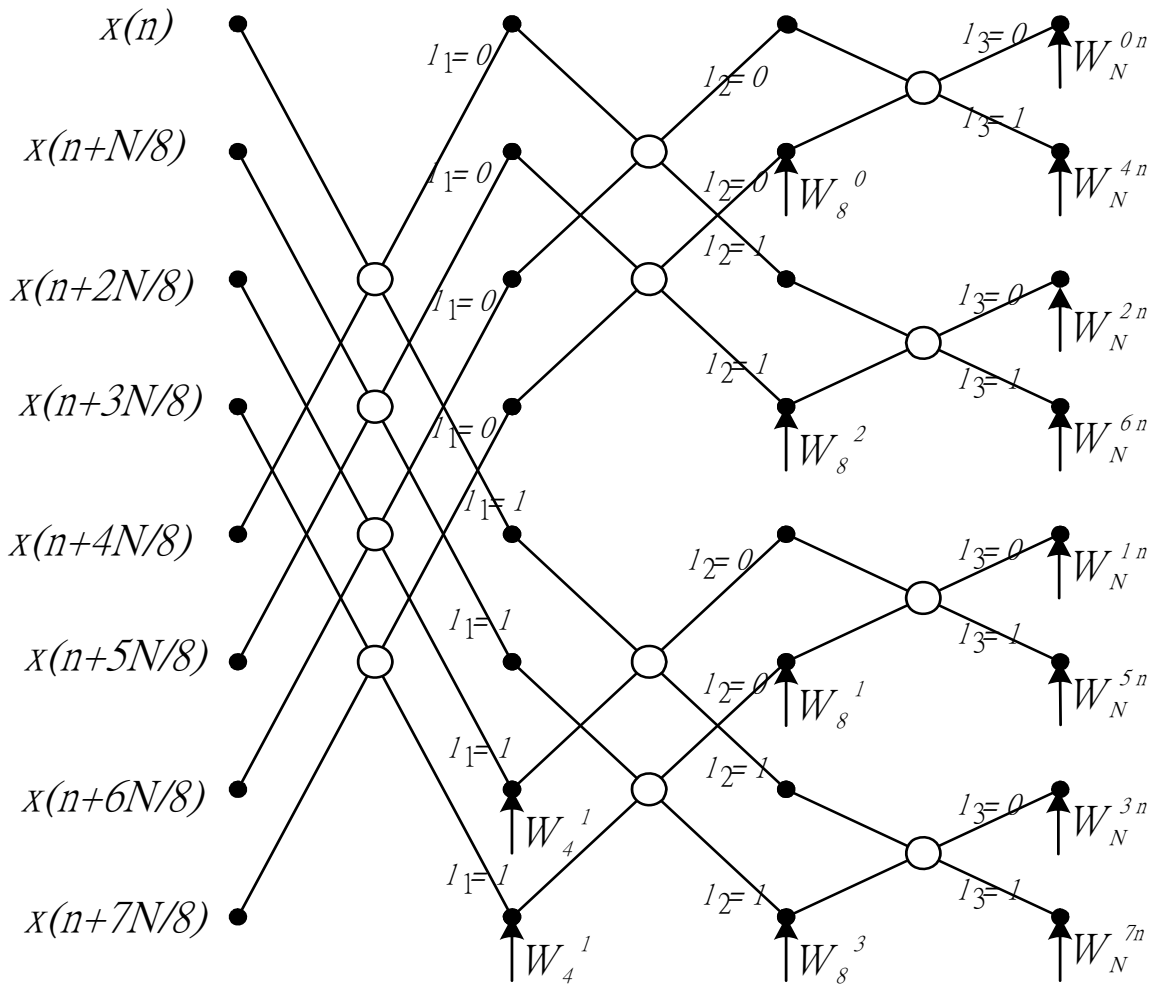
FFT Algorithms

- Butterfly of Radix-8 Algorithm



FFT Algorithms

- Butterfly of Radix-2^3 Algorithm



DIF Split-Radix 2/4 Algorithm

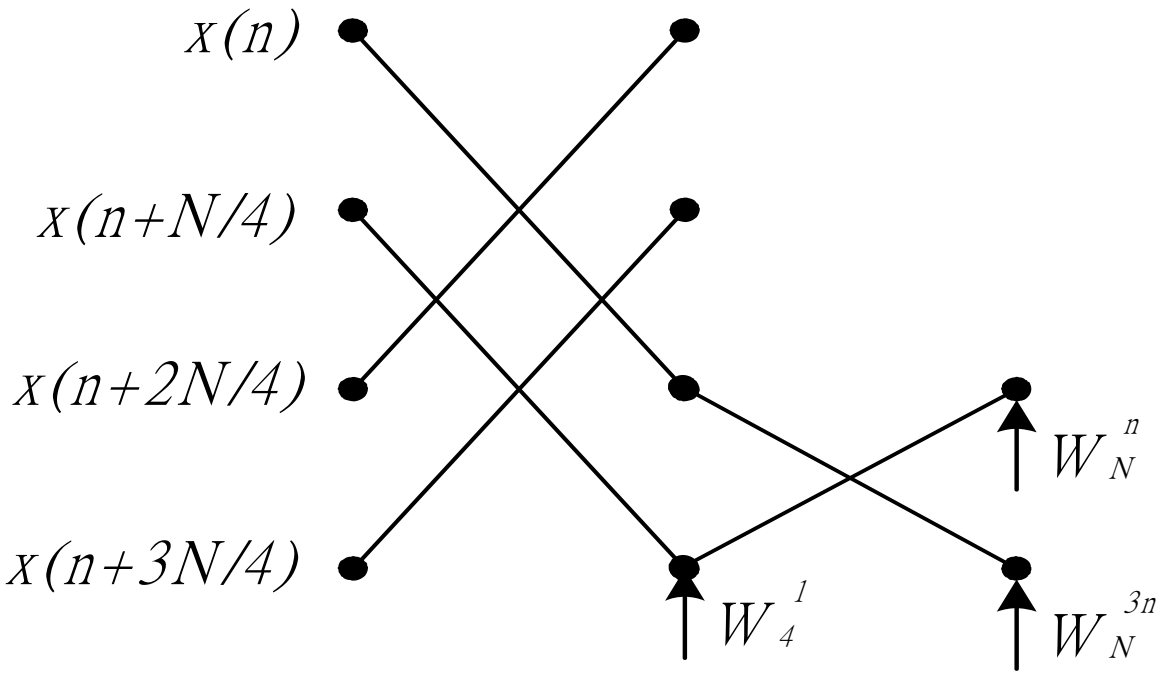
$$\begin{cases} X(2k) = \sum_{n=0}^{N/2-1} [x(n) + x(n+\frac{2N}{4})]W_{N/2}^{nk} \\ X(4k+1) = \sum_{n=0}^{N/4-1} \{x(n) - x(n+\frac{2N}{4}) - j[x(n+\frac{N}{4}) - x(n+\frac{3N}{4})]\}W_N^n W_N^{4nk} \\ X(4k+3) = \sum_{n=0}^{N/4-1} \{x(n) - x(n+\frac{2N}{4}) + j[x(n+\frac{N}{4}) - x(n+\frac{3N}{4})]\}W_N^{3n}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

k in *X(2k)* is from *0* to *N/2-1*,

and in *X(4k+1)* and *X(4k+3)* are from *0* to *N/4-1*

FFT Algorithms

- Butterfly of Split-Radix 2/4 Algorithm



FFT Algorithms

- Advantage of Radix-2/4 Algorithm
  - Low Computational Complexity
  - Flexible as radix-2 algorithm
  - Bit reversed output (when normally ordered input)

FFT Algorithms

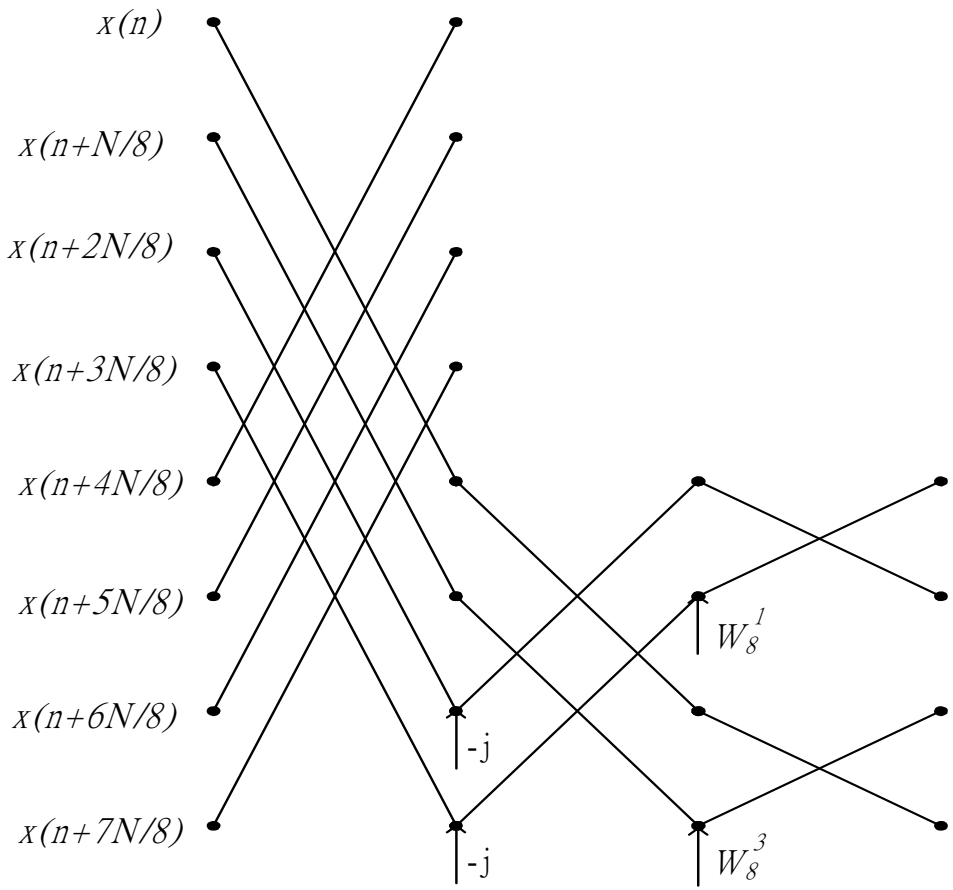
- DIF Split-Radix 2/8 Algorithm

$$\begin{cases} X(2k) = \sum_{n=0}^{N/2-1} [x(n) + x(n+\frac{2N}{4})]W_N^{2nk} \\ X(8k+l) = \sum_{n=0}^{N/8-1} \{[x(n) + x(n+\frac{2N}{8})W_4^l + x(n+\frac{4N}{8})W_4^{2l} + x(n+\frac{6N}{8})W_4^{-l}] \\ \qquad + [x(n+\frac{N}{8}) + x(n+\frac{3N}{8})W_4^l + x(n+\frac{5N}{8})W_4^{2l} + x(n+\frac{7N}{8})W_4^{-l}]W_8^l\}W_N^{nl}W_{N/8}^{nk} \end{cases}$$

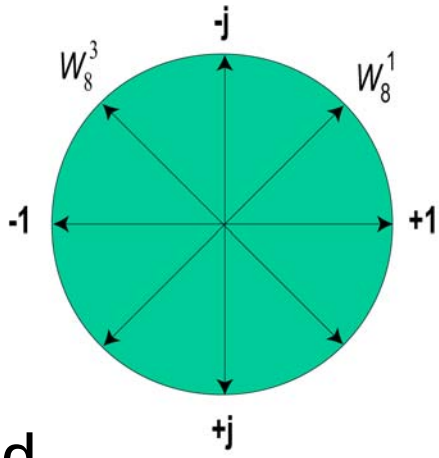
$$l = 1,3,5,7$$

FFT Algorithms

- Butterfly of Split-Radix 2/8 Algorithm

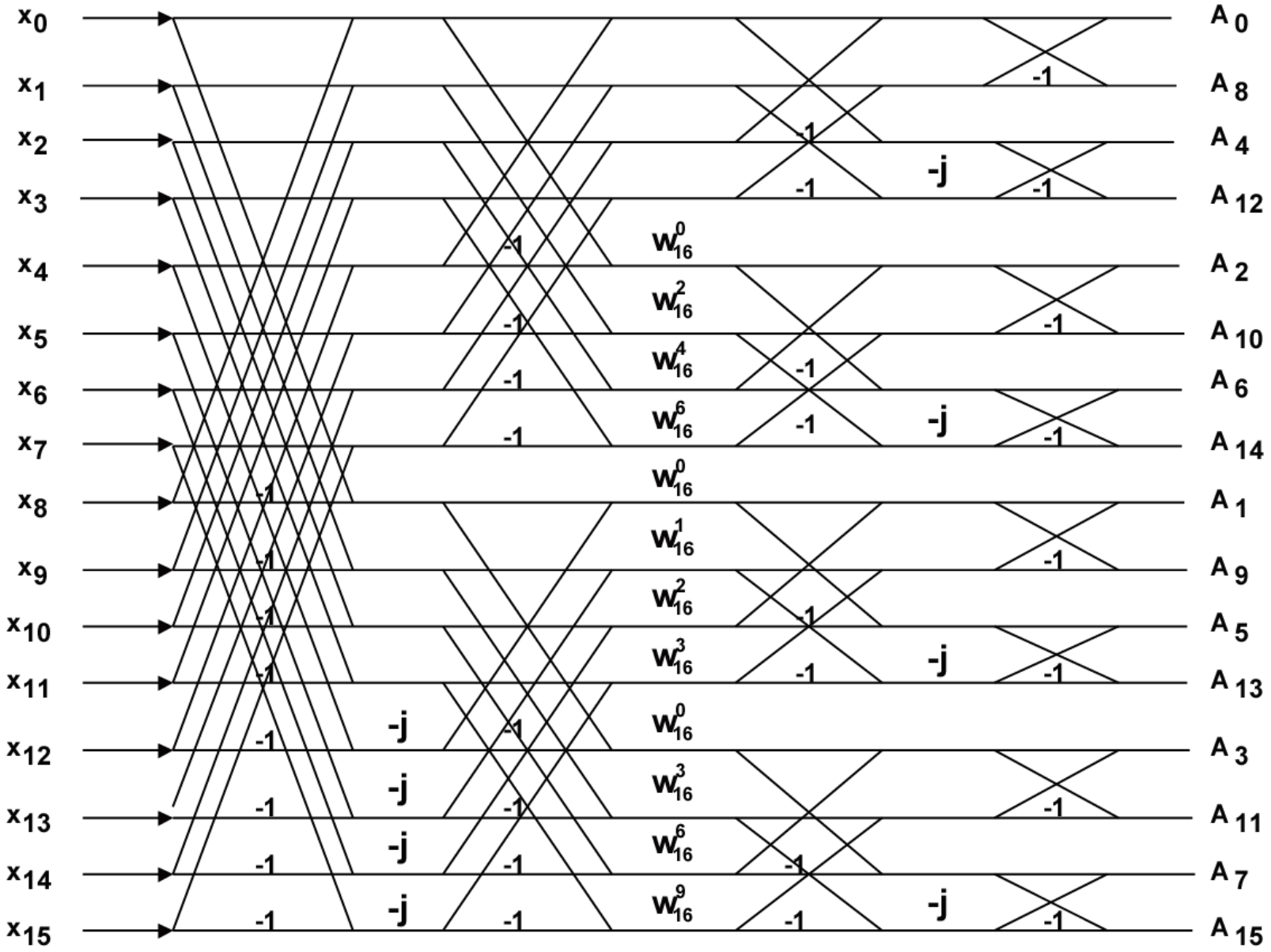


Multiplicative Complexity

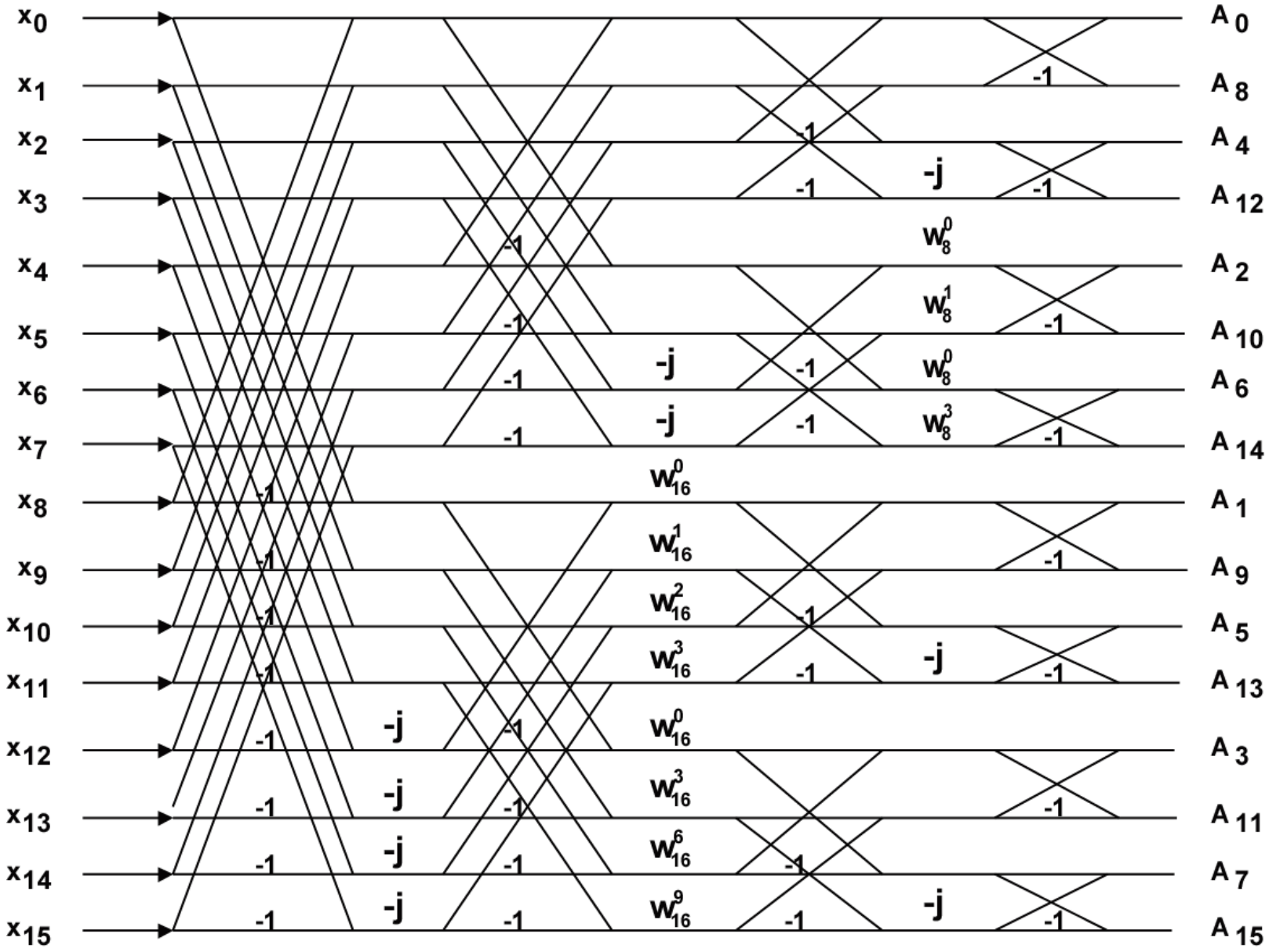


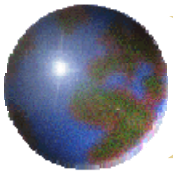
- Trivial multiplications in FFT
  - Multiplied by
    - Radix-2: ±1 removed
    - Radix-4: ±1 and ±j (partially) removed
    - Split-radix(2/4): ±1 and ±j removed
    - Radix-8: ±1, ±j, $(1\pm j)/\sqrt{2}$ (partially) removed
    - Radix-2/8: ±1, ±j, $(1\pm j)/\sqrt{2}$ removed

Radix-4 Signal Flow Graph



Split-Radix Signal Flow Graph





Multiplicative Complexity

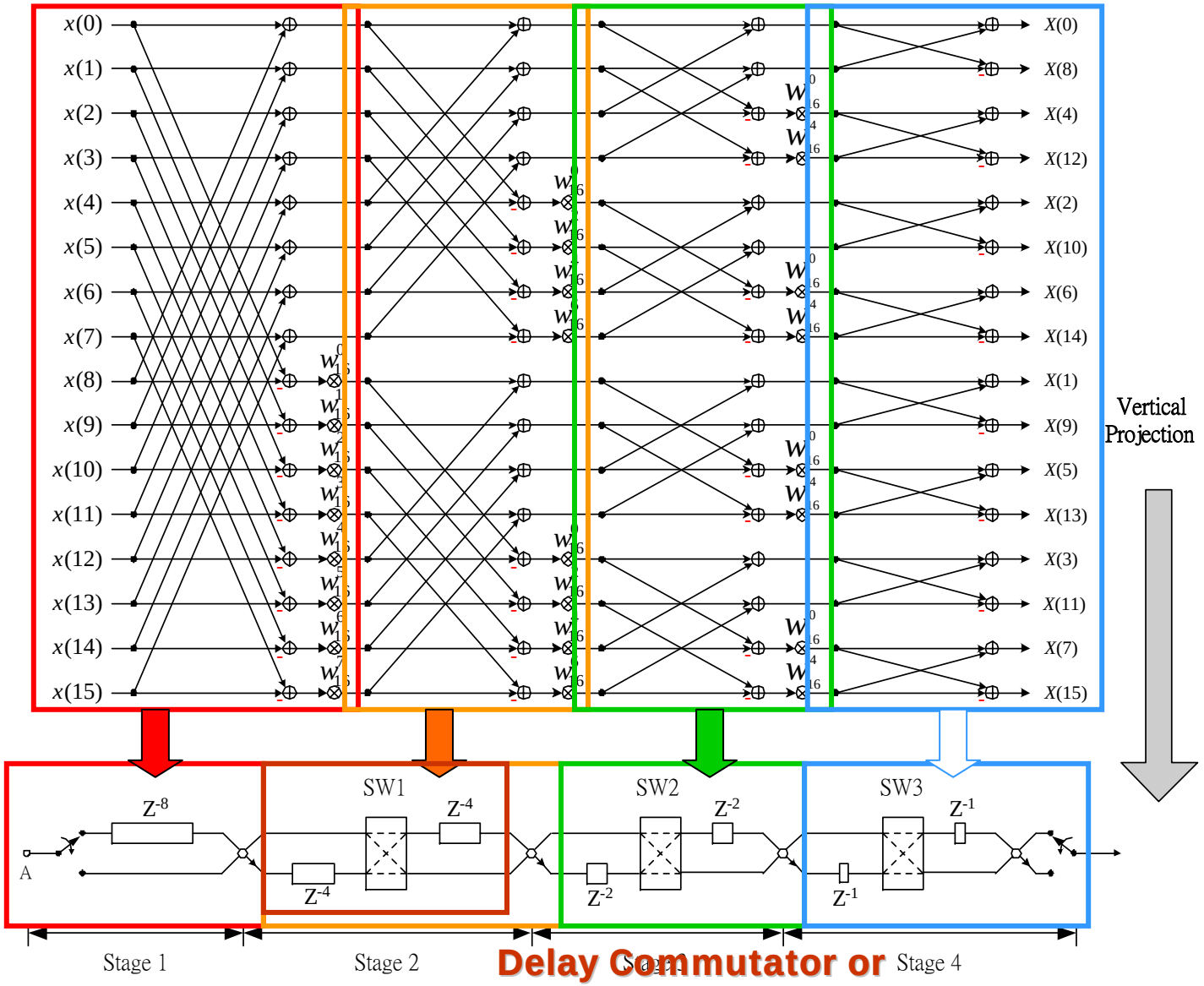
N	Radix-2	Radix-4	Split-Radix	Radix-8	Const. Mul	Radix-2/8	Const. Mul
8	2	3	2	0	2	0	2
16	10	8	8	6	4	4	6
32	34	31	26	20	8	16	14
64	98	76	72	48	32	44	38
128	258	215	186	152	64	120	94
256	642	492	456	376	128	308	214
512	1538	1239	1082	824	384	736	494
1024	3586	2732	2504	2104	768	1724	1126
2048	8194	6487	5690	4792	1536	3976	2494
4096	18434	13996	12744	10168	4096	8964	5494
8192	40962	32087	28218	23992	8192	19952	12046

How to obtain regular SR FFT architecture?

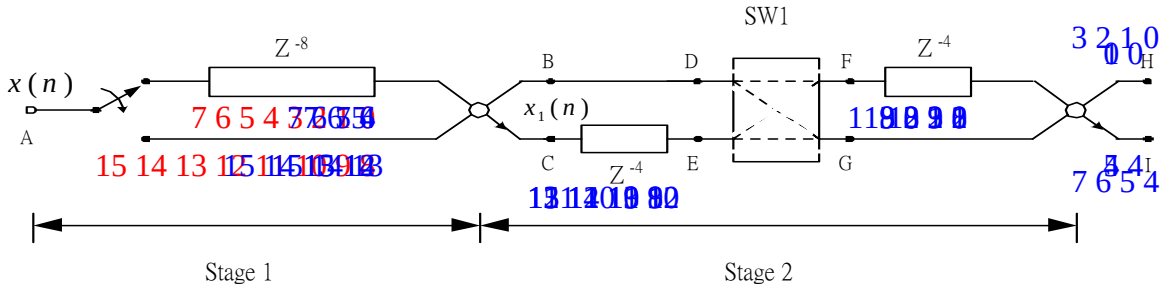
Architecture Level

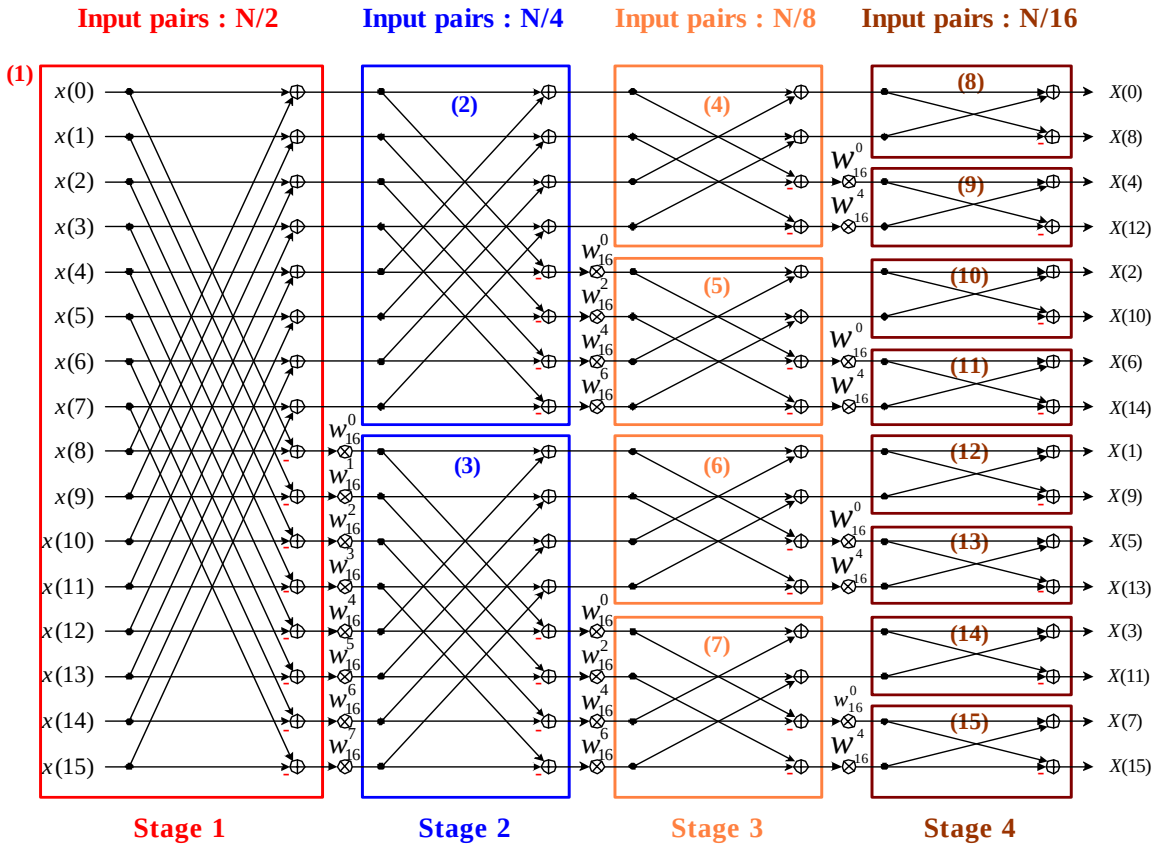
- Mapping procedure
  - Systolic array techniques
    - Operation scheduling, resource sharing
  - Pipeline architecture
    - One-dimensional linear array
    - Delay-feedback vs. Delay-commutator.
  - Single PE architecture
    - Shared-memory, Single Processing Element (PE)

R2MDC Radix-2 Multi-Path Delay Commutator

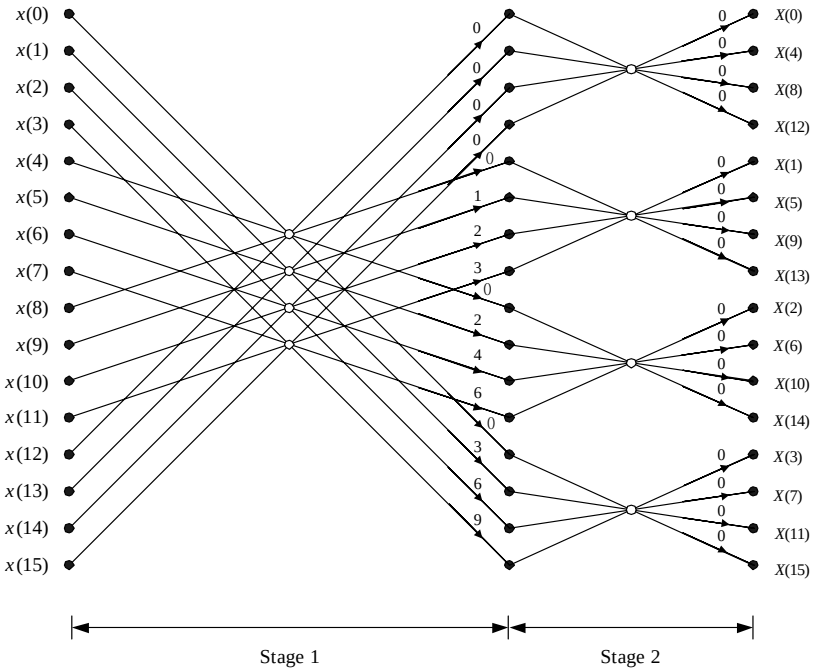


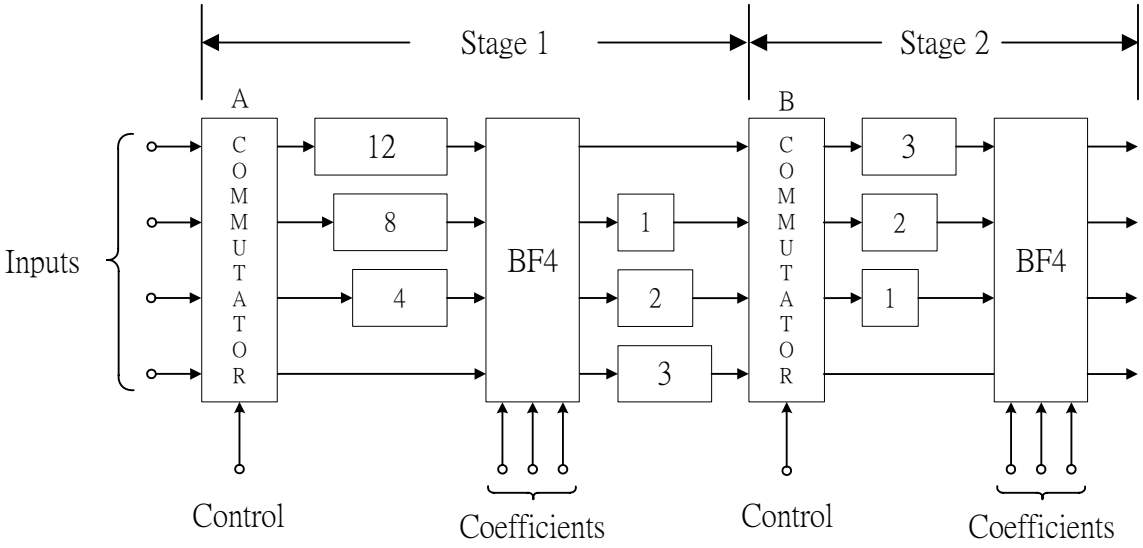
1st and 2nd stages in R2MDC (N=16)

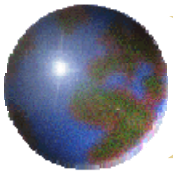




R4MDC Radix-4 Multi-Path Delay Commutator

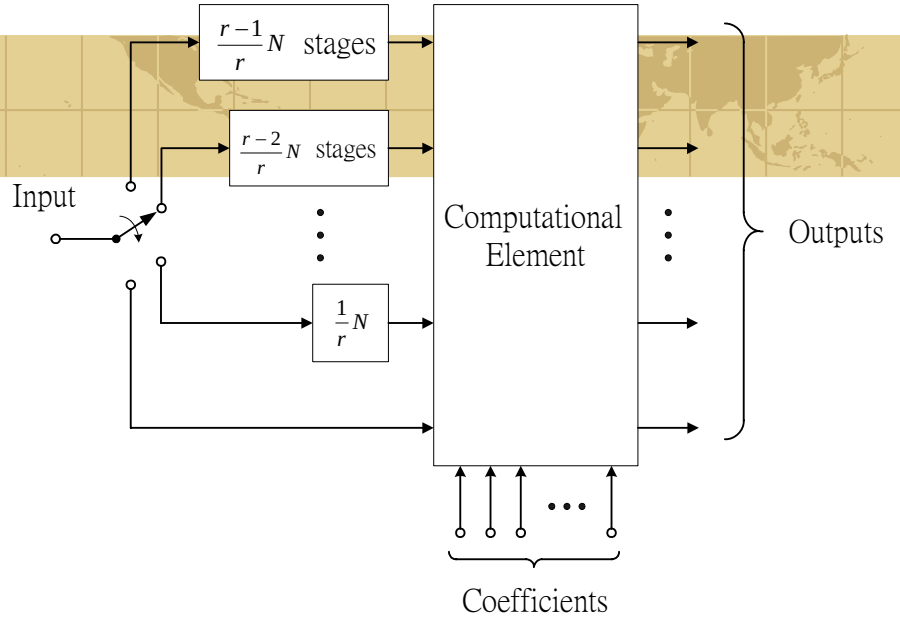






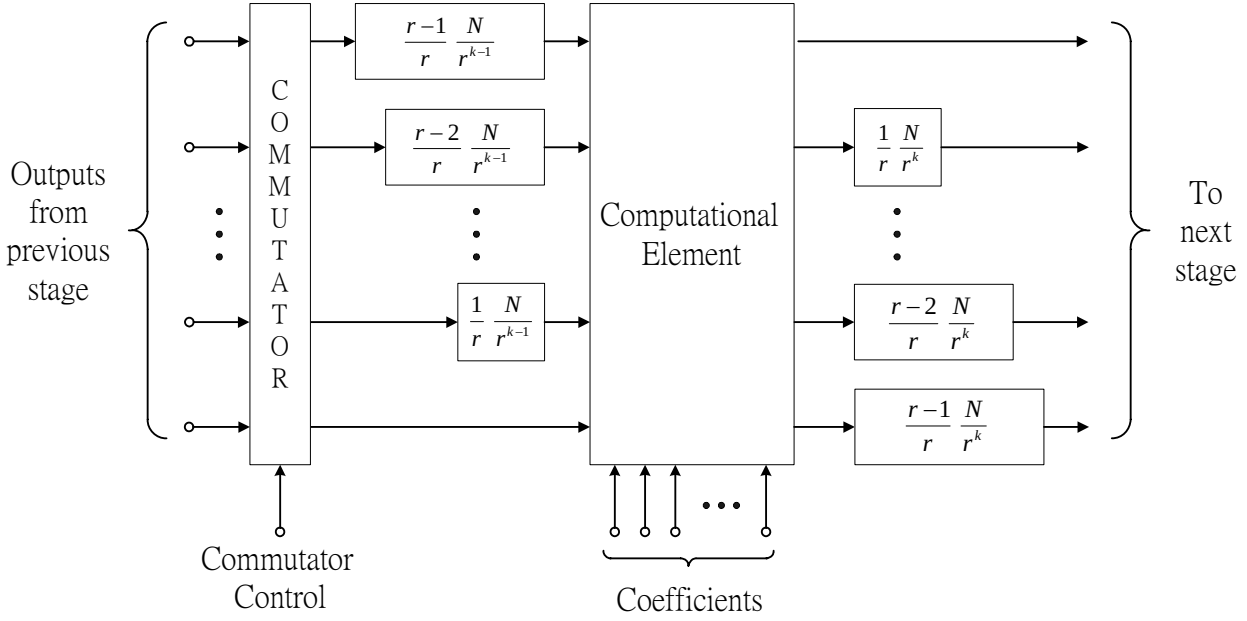
RrMDC

Input stage

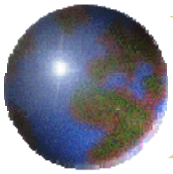


(a)

k th stage



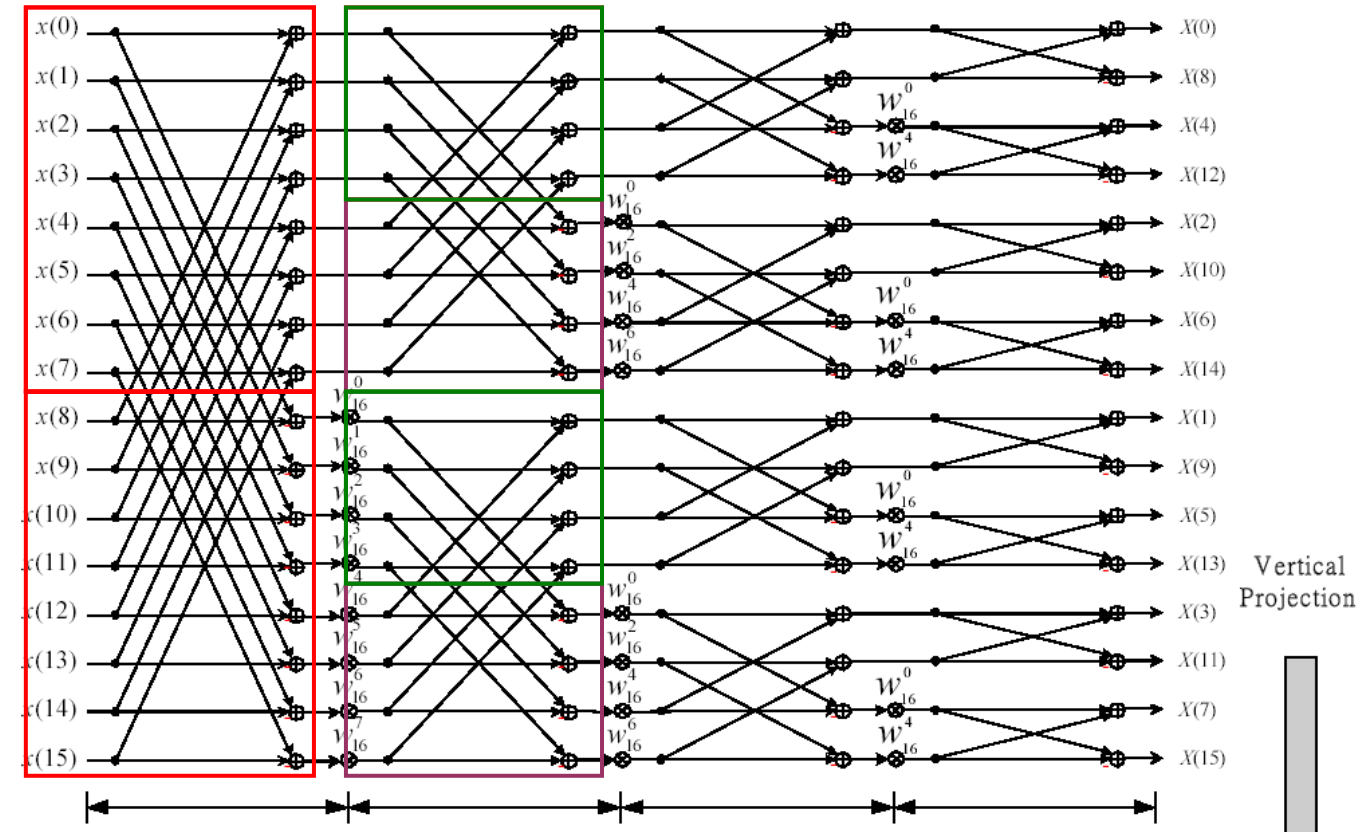
(b)



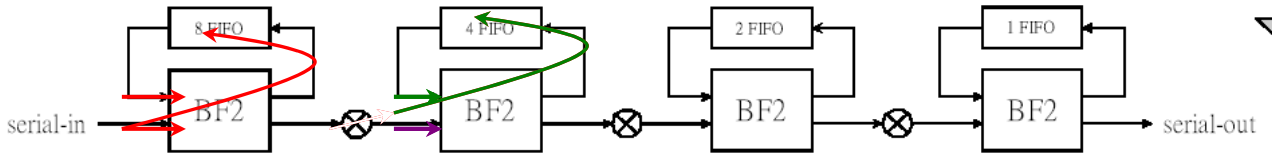
Delay Feedback

- R2SDF
- R4SDF
- $R2^2SDF$

R2SDF(N=16) Radix-2 Single-Path Delay Feedback

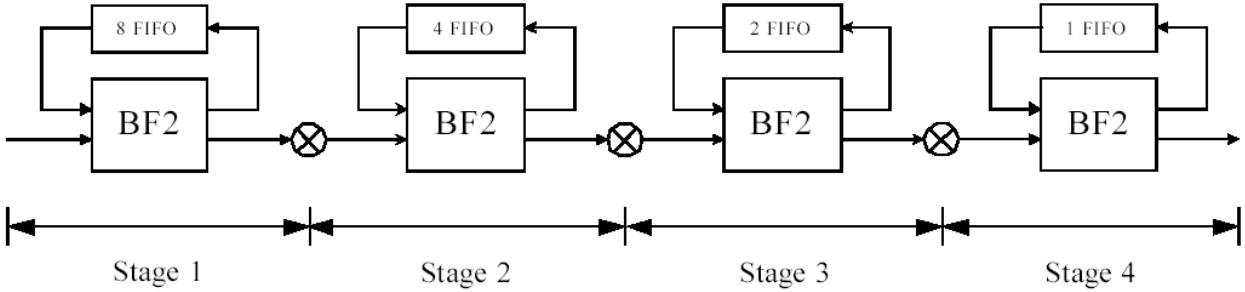


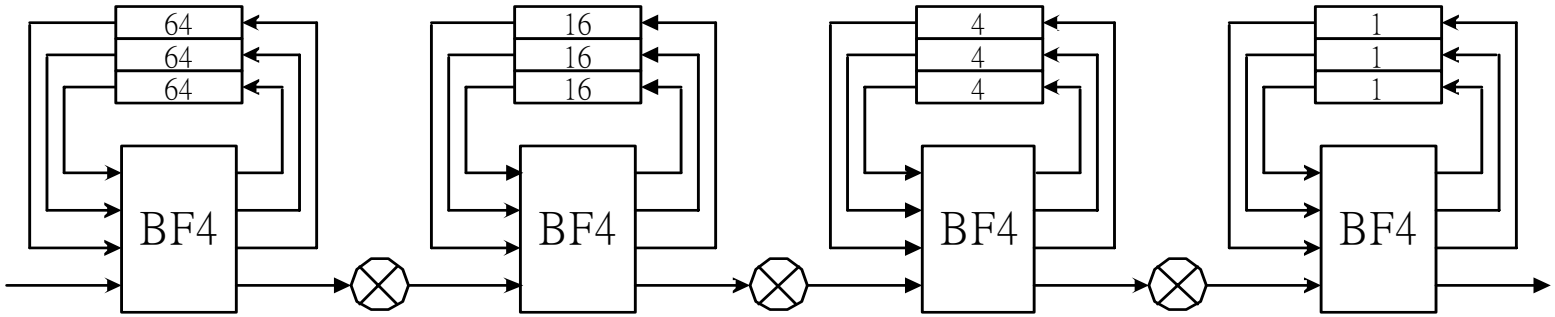
(a)



(b)

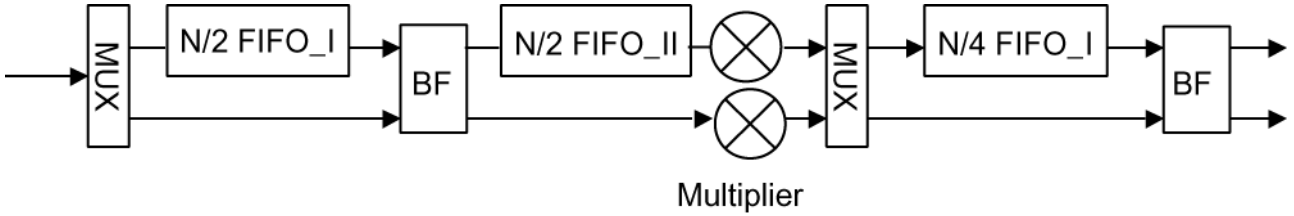
R2SDF (N=16) *vs. R4SDF* (N=128)



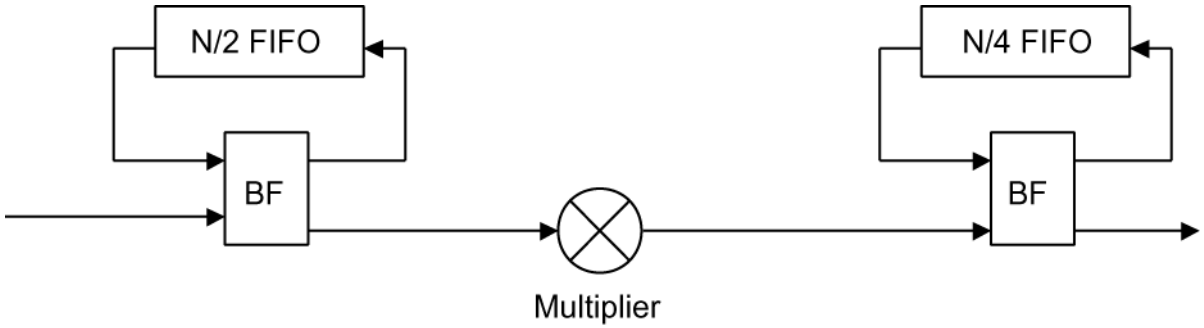


Buffer Styles of pipeline architecture

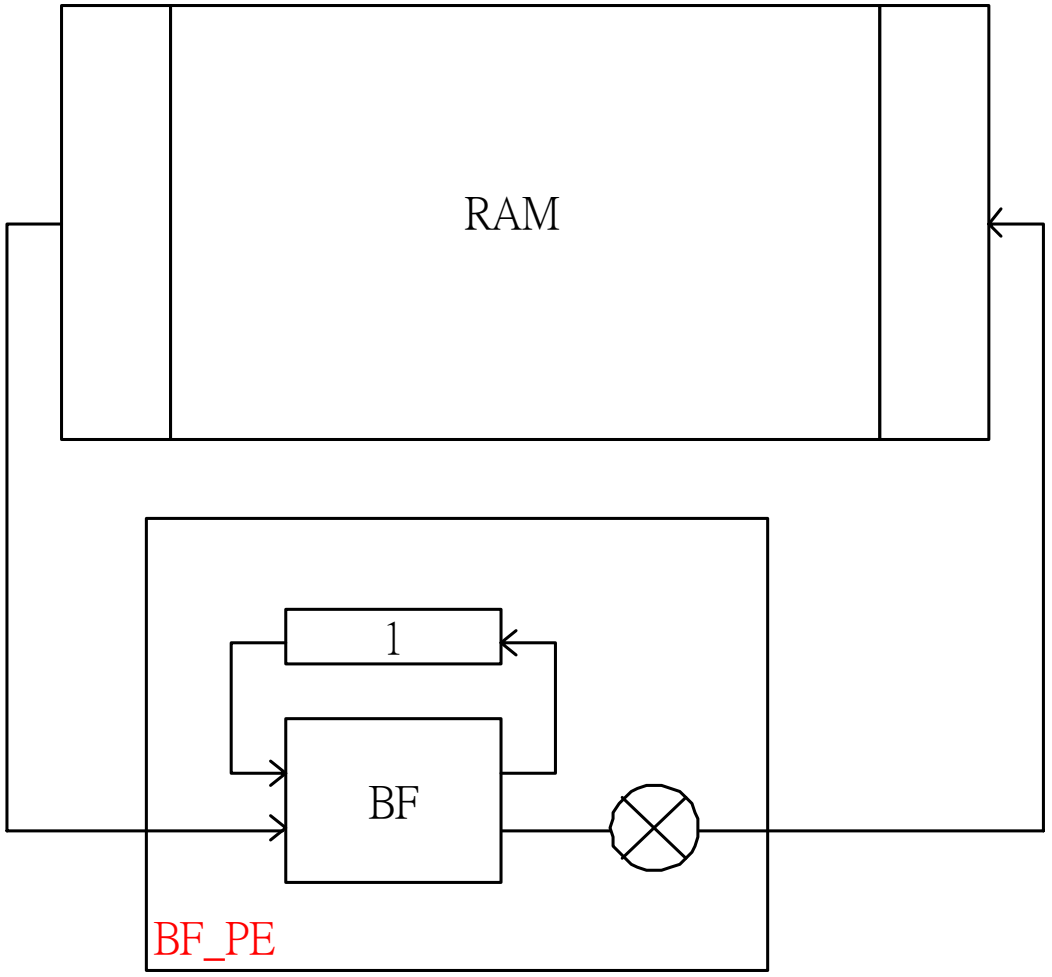
- R2 delay-commutator: inefficient (50%) MEM usage. (R2MDC)



- R2 delay-feedback: 100% MEM usage.(R2SDF)



Single PE Architecture



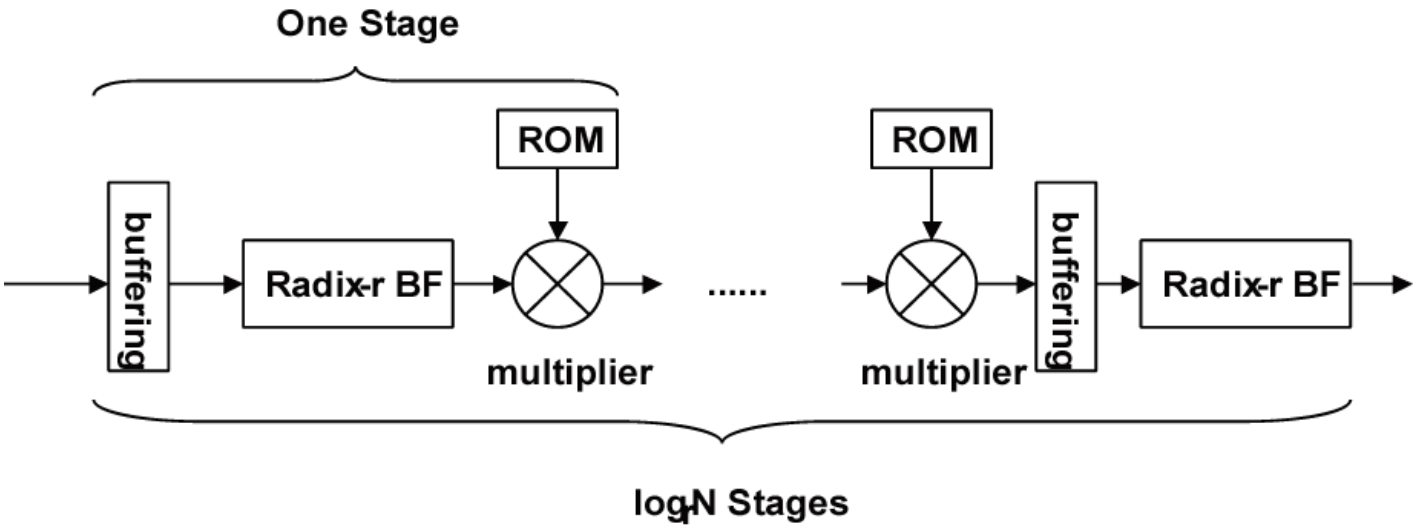
single BF_PE radix-2 shared memory architecture

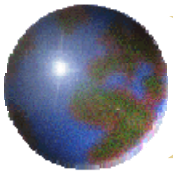
Concluding Remarks

- The Split-Radix algorithm has less computation complexity, comparing with the fixed Radix algorithm. However, its butterfly operation is irregular (L-shape).
- The processing speed of pipeline architecture is faster than single-PE architecture. However, the single PE architecture is the most area-efficient, especially for long length FFT/IFFT application.

Review Traditional FFT Design

- Steps
  1. Given N-point FFT spec., choose fixed-radix algorithm
  2. Design radix-r butterfly, multiplier, etc.
  3. Cascade $\log_r N$ stages to compute N point FFT.
- Arbitrary radix can be used
  - Base on Cooley-Tukey decomposition for any composite number





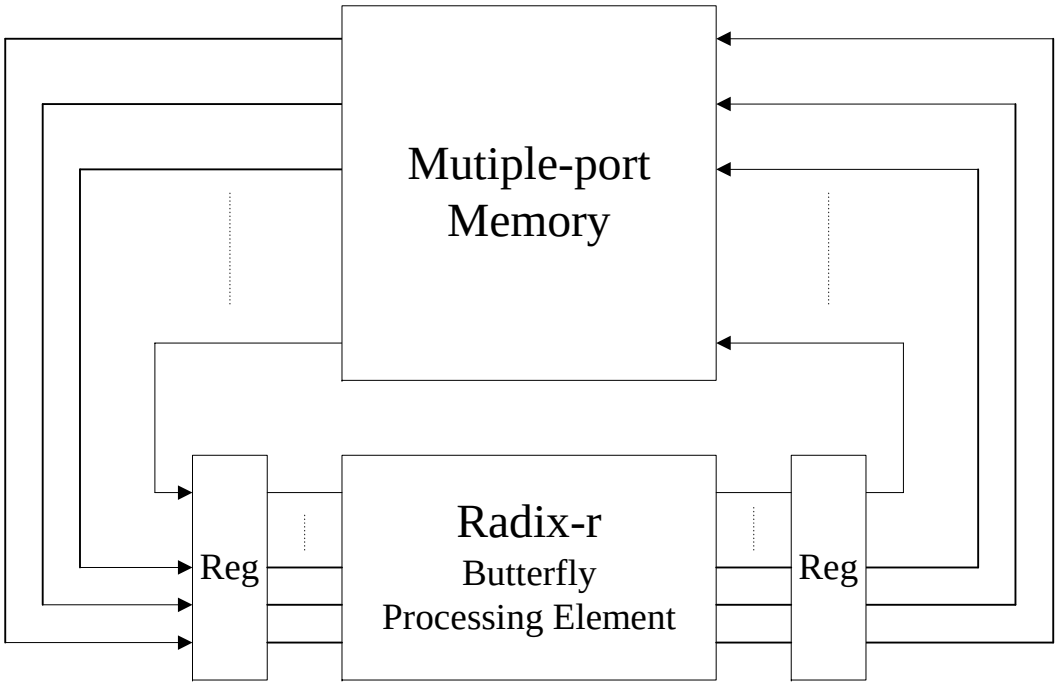
Problem of Traditional Approach

- Cannot drive architecture for mixed-Radix algorithm
- The processing speed is no longer the critical issue any more nowadays.
- The chip area and the power consumption dominate the design quality.
- Re-configurable FFT/IFFT architecture design is necessary for various applications.

☞ A length-scalable and latency-specified FFT/IFFT core is necessary.

Proposed Solution

- We implement FFT module by single PE architecture



Pre-fetch buffer

Design Issue

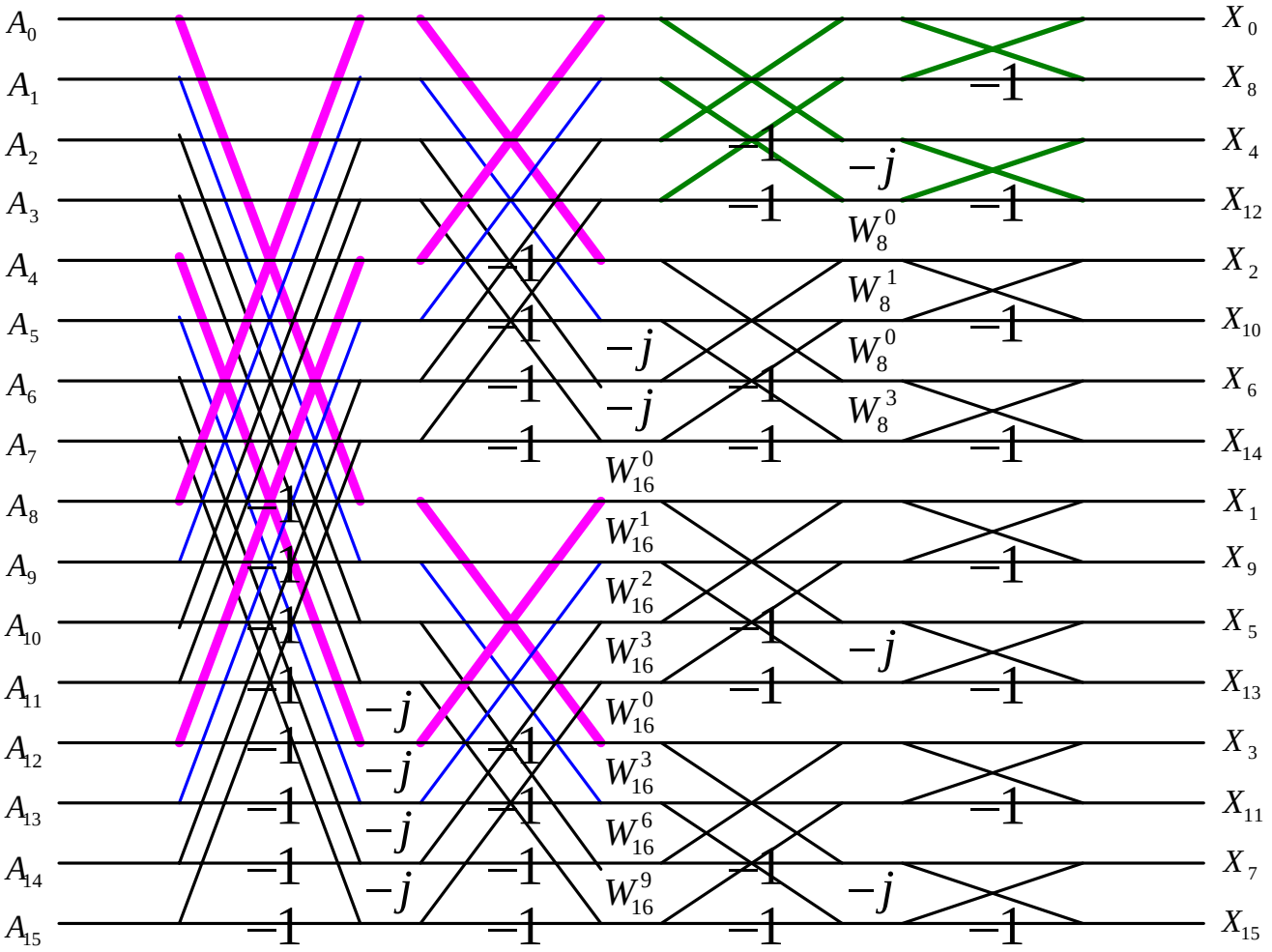
- Performance-enough, Chip area, power consumption.
- Scalable processing element.
- Limited Storage block(s).
- Efficient memory address generator.

Algorithm Level

- We adopt split-radix 2/4 algorithm to realize the FFT module.

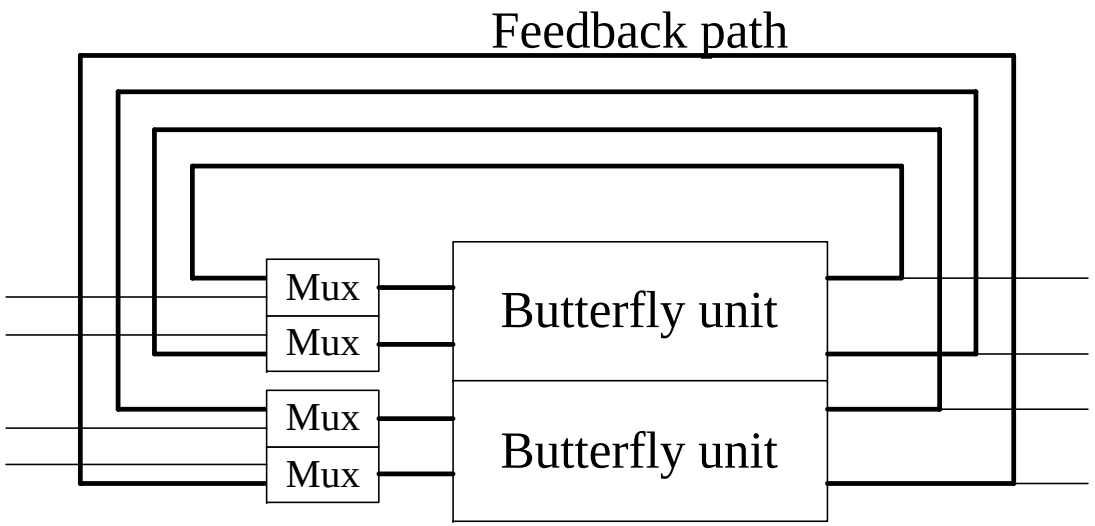
$$\begin{cases} \boldsymbol{A_{2k}} = \sum_{n=0}^{N/2-1} (X_n + X_{n+N/2}) \cdot W_{N/2}^{n \cdot k} \\ \boldsymbol{A_{4k+1}} = \sum_{n=0}^{N/4-1} (X_n - j \cdot X_{n+N/4} - X_{n+N/2} + j \cdot X_{n+3N/4}) \cdot W_N^n \cdot W_{N/4}^{n \cdot k} \\ \boldsymbol{A_{4k+3}} = \sum_{n=0}^{N/4-1} (X_n + j \cdot X_{n+N/4} - X_{n+N/2} - j \cdot X_{n+3N/4}) \cdot W_N^{3n} \cdot W_{N/4}^{n \cdot k} \end{cases}$$

The Kernel of Processing Element



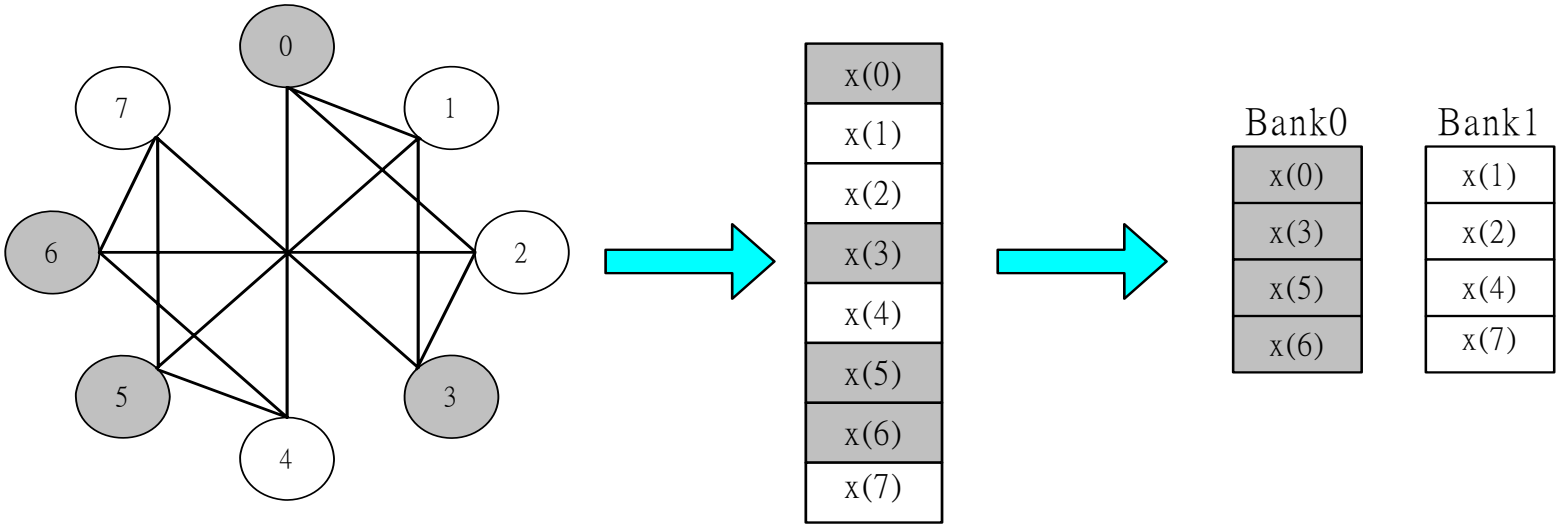
Folded Butterfly Units

- Comparing with Radix-2/Radix-2^2, it saves half memory access times.



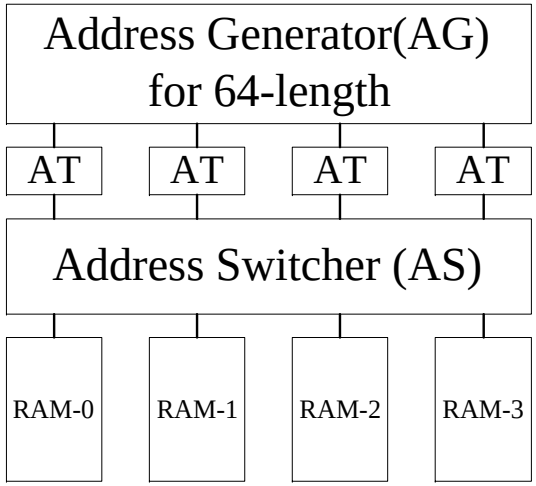
Storage Blocks

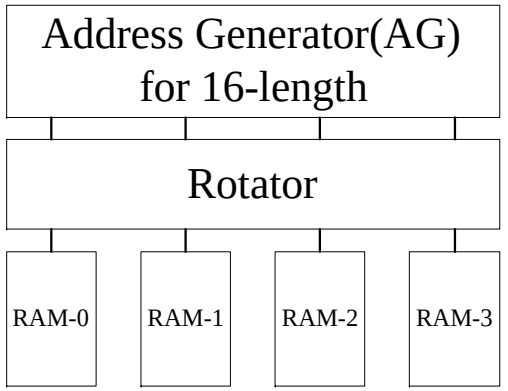
- We use multiple single-port memory banks to replace the multi-port memory.
- The concept of conflict-free memory. (Vertex coloring problem)



Scalable Memory Address Generator

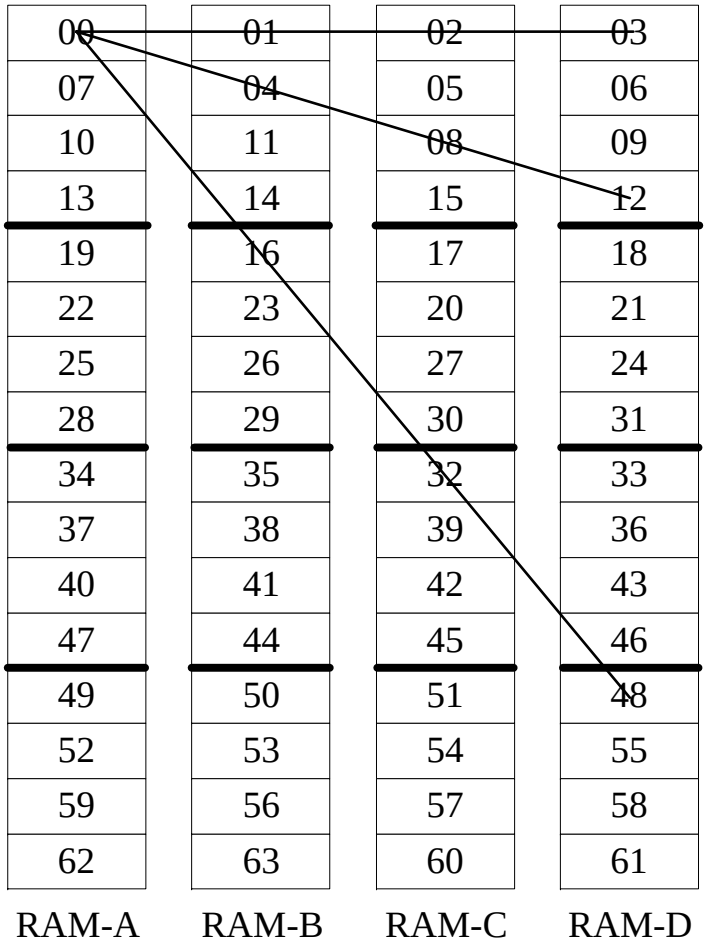
- There must exist a solution for such vertex coloring problem.
- The best solution --- The proposed Interleave Rotated Data Allocation (IRDA) algorithm.



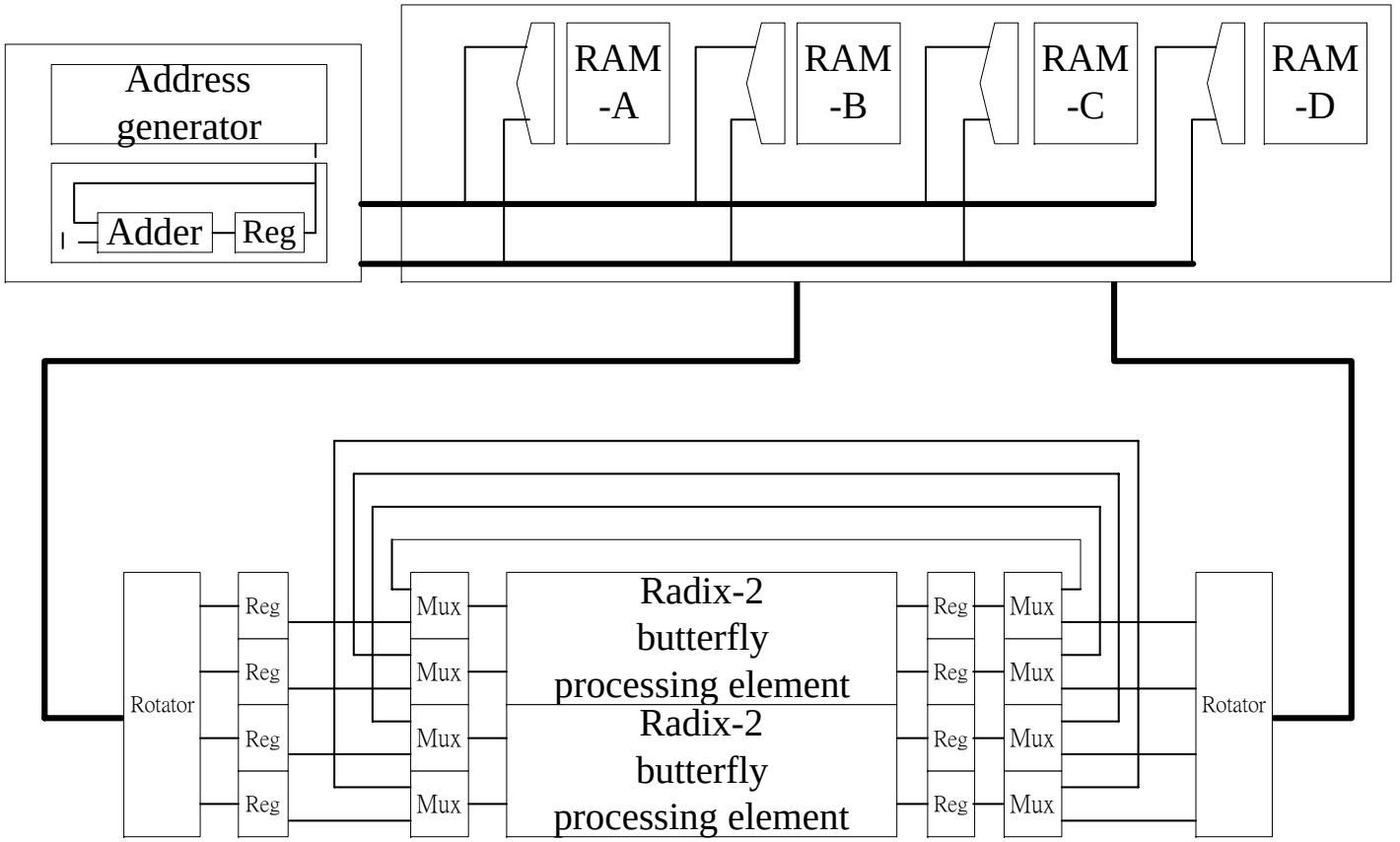


The IRDA Concept

- A conflict-free memory banks.
- Simple and length-scalable design.
- The circular shift rotator.

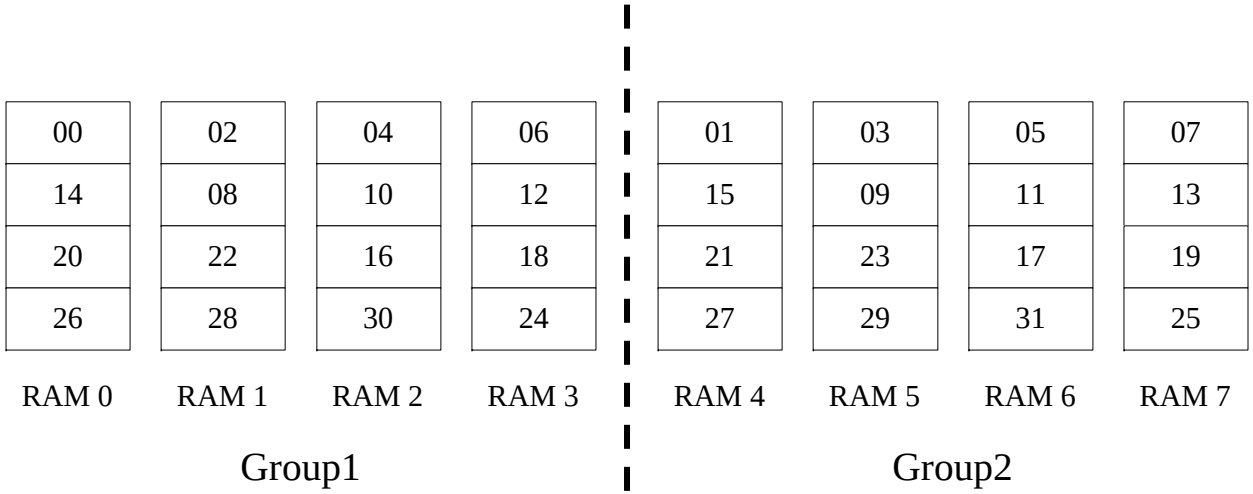


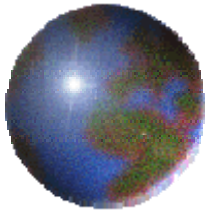
Length-Scalable FFT/IFFT Core



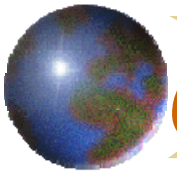
Further Performance Improvement

- Multiple PEs architecture.
- 2 pipeline PEs, for example.



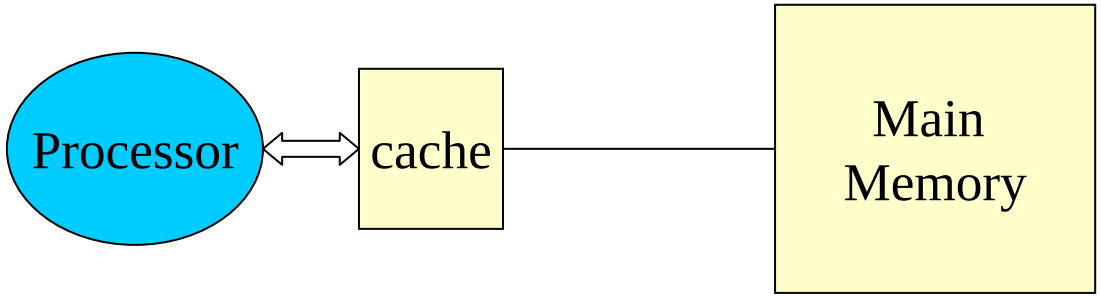


The Cached-FFT Algorithm

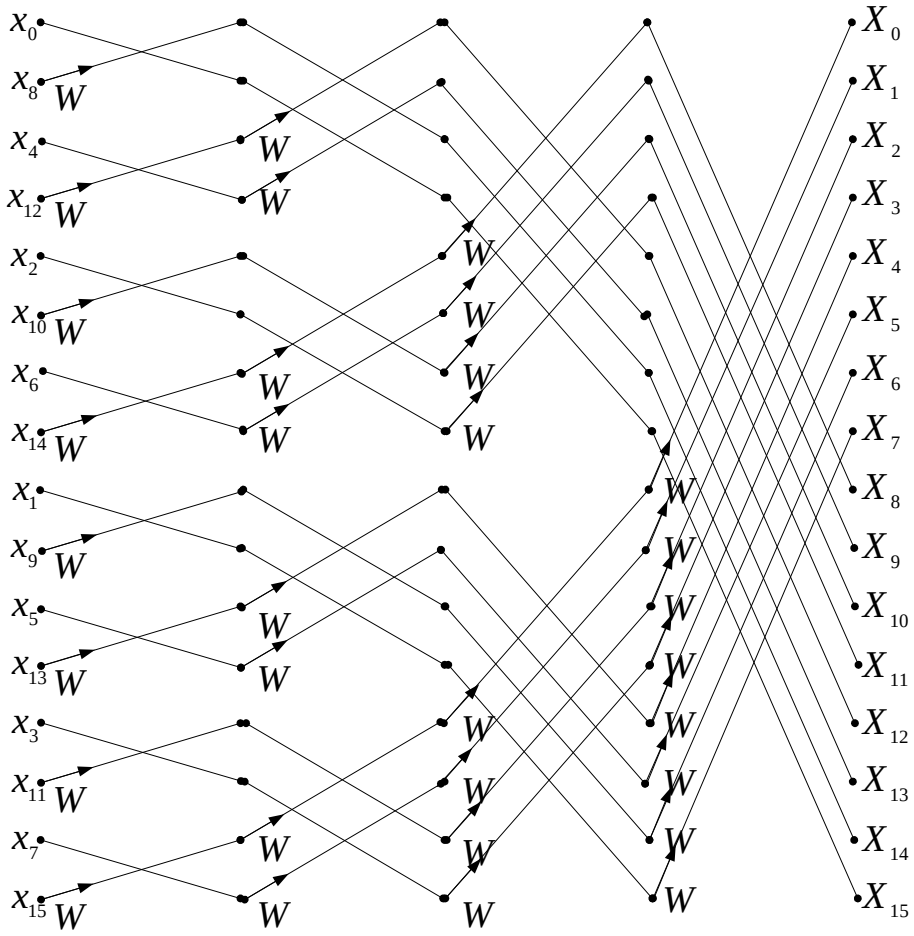


Overview

1. Input data are loaded into an *N*-word main memory.
2. *C* of the *N* words are loaded into the cache.
3. As many butterflies as possible are computed using the data in the cache.
4. Processed data in the cache are flushed to main memory.
5. Steps 2-4 are repeated until all *N* words have been processed once.
6. Steps 2-5 are repeated until the FFT has been completed.



Result


x0
x8
x4
x12
x2
x10
x6
x14
x1
x9
x5
x13
x3
x11
x7
x15
W
X0
X1
X2
X3
X4
X5
X6
X7
X8
X9
X10
X11
X12
X13
X14
X15

N=64, E=2, Radix-2 Cached-FFT