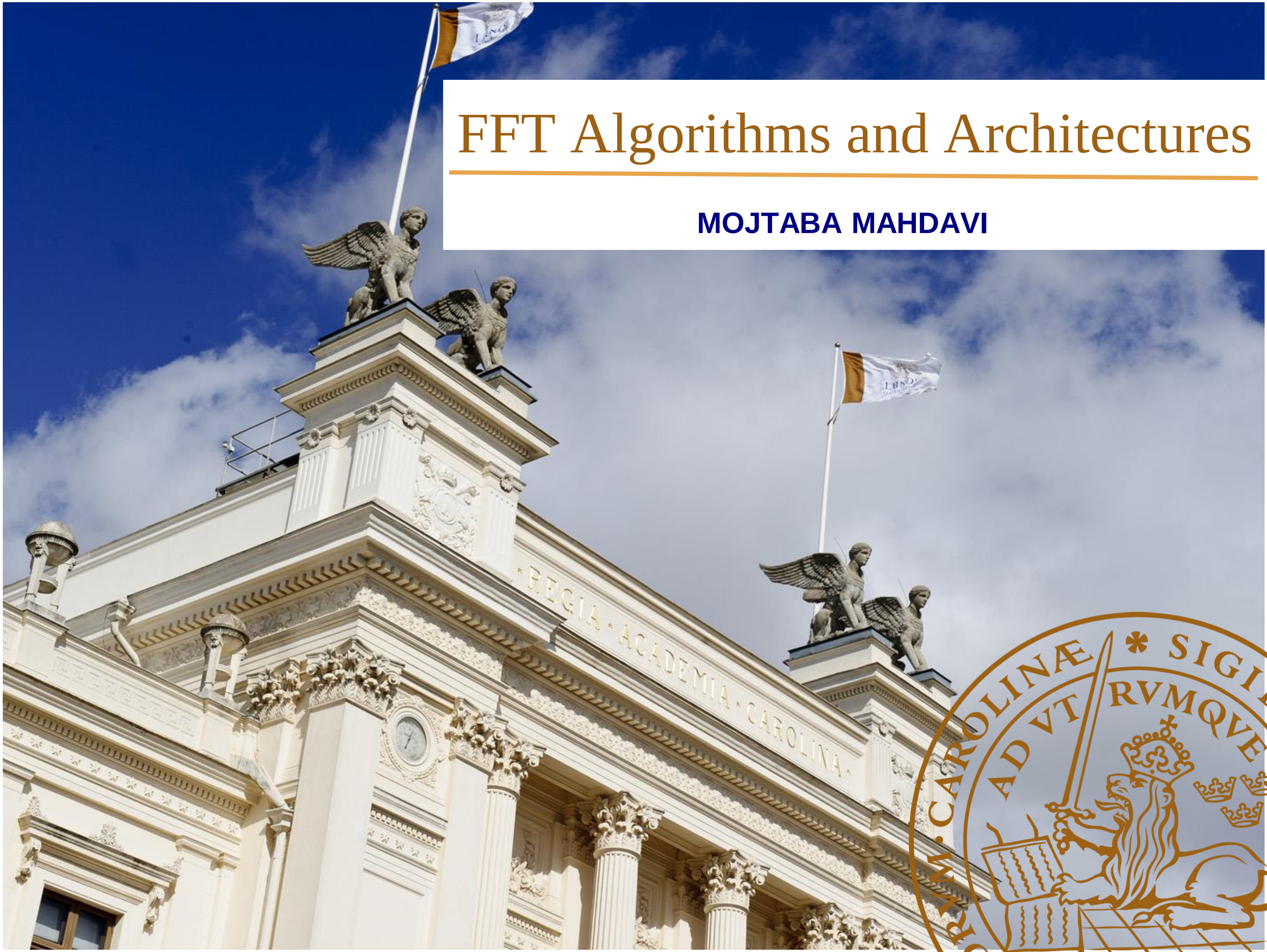
FFT Algorithms and Architectures

MOJTABA MAHDAVI



Outline

- Discrete Fourier Transform (DFT)
- Fast Fourier Transform (FFT)
- Twiddle Factor Multiplication
- FFT Algorithms
- FFT Architectures
- Data Flow Processing
- DIF vs. DIT Decomposition

Outline

- Discrete Fourier Transform (DFT)
- Fast Fourier Transform (FFT)
- Twiddle Factor Multiplication
- FFT Algorithms
- FFT Architectures
- Data Flow Processing
- DIF vs. DIT Decomposition

Discrete Fourier Transform (DFT)

- DFT is one of the most important algorithms in Digital Signal Processing (DSP).

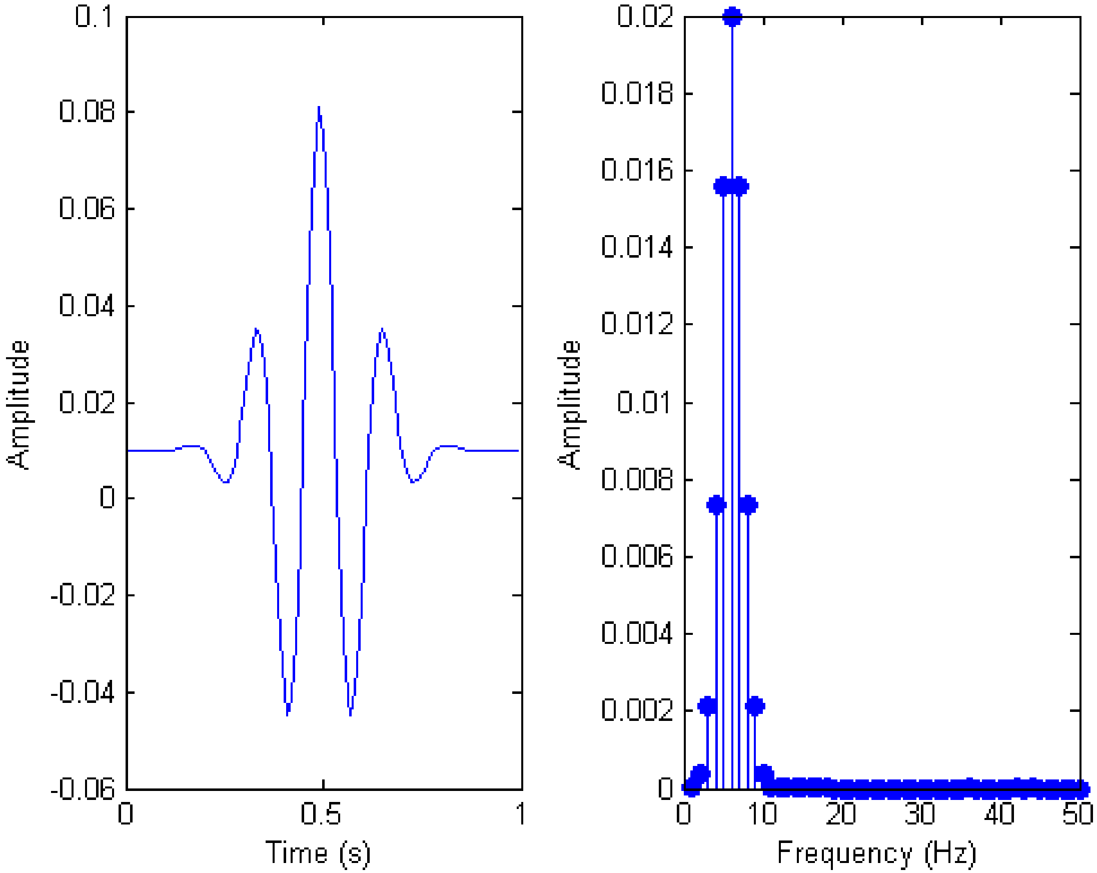
- DFT is widely used in several applications:
  - Audio and Image Processing
  - Spectrum Analysis of Signals
  - Digital Communication Transmitter/Receivers

Spectrum Analysis

- DFT calculates the frequency spectrum of a signal (discrete sinusoids components) to examine the information encoded in:
  - Frequency
  - Phase
  - Amplitude

- DFT can find a system's frequency response from the system's impulse response and vice versa
  - Analyze the frequency/time-domain behavior of a system

Spectrum Analysis



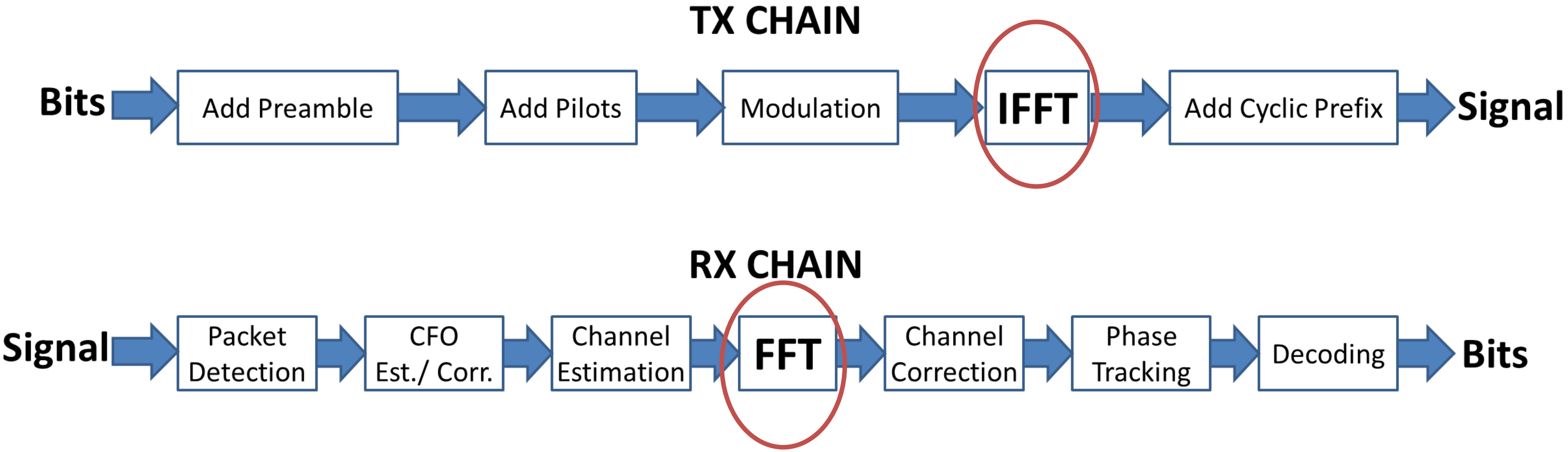
Digital Communication Transmitter/Receiver

- DFT is extensively used in multi-carrier transmission systems like orthogonal frequency domain multiplexing (OFDM).

- DFT and IDFT are used to perform OFDM demodulation and modulation, respectively.

Digital Communication Transmitter/Receiver

- Simple OFDM physical layer chain.



DFT

- An *N*-point DFT is calculated as:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \qquad k = 0, 1, 2, ..., N-1$$

- Twiddle factors:

$$W_N^{nk} = e^{-j2\pi nk/N} = \cos(2\pi nk/N) - j\sin(2\pi nk/N)$$

- Complexity: $\mathcal{O}(N^2)$

8-point DFT

$X(0) =$	$x(0)W_8^0$	$+ x(1)W_8^0$	$+ x(2)W_8^0$	$+ x(3)W_8^0$	$+ x(4)W_8^0$	$+ x(5)W_8^0$	$+ x(6)W_8^0$	$+ x(7)W_8^0$
$X(1) =$	$x(0)W_8^0$	$+ x(1)W_8^1$	$+ x(2)W_8^2$	$+ x(3)W_8^3$	$+ x(4)W_8^4$	$+ x(5)W_8^5$	$+ x(6)W_8^6$	$+ x(7)W_8^7$
$X(2) =$	$x(0)W_8^0$	$+ x(1)W_8^2$	$+ x(2)W_8^4$	$+ x(3)W_8^6$	$+ x(4)W_8^8$	$+ x(5)W_8^{10}$	$+ x(6)W_8^{12}$	$+ x(7)W_8^{14}$
$X(3) =$	$x(0)W_8^0$	$+ x(1)W_8^3$	$+ x(2)W_8^6$	$+ x(3)W_8^9$	$+ x(4)W_8^{12}$	$+ x(5)W_8^{15}$	$+ x(6)W_8^{18}$	$+ x(7)W_8^{21}$
$X(4) =$	$x(0)W_8^0$	$+ x(1)W_8^4$	$+ x(2)W_8^8$	$+ x(3)W_8^{12}$	$+ x(4)W_8^{16}$	$+ x(5)W_8^{20}$	$+ x(6)W_8^{24}$	$+ x(7)W_8^{28}$
$X(5) =$	$x(0)W_8^0$	$+ x(1)W_8^5$	$+ x(2)W_8^{10}$	$+ x(3)W_8^{15}$	$+ x(4)W_8^{20}$	$+ x(5)W_8^{25}$	$+ x(6)W_8^{30}$	$+ x(7)W_8^{35}$
$X(6) =$	$x(0)W_8^0$	$+ x(1)W_8^6$	$+ x(2)W_8^{12}$	$+ x(3)W_8^{18}$	$+ x(4)W_8^{24}$	$+ x(5)W_8^{30}$	$+ x(6)W_8^{36}$	$+ x(7)W_8^{42}$
$X(7) =$	$x(0)W_8^0$	$+ x(1)W_8^7$	$+ x(2)W_8^{14}$	$+ x(3)W_8^{21}$	$+ x(4)W_8^{28}$	$+ x(5)W_8^{35}$	$+ x(6)W_8^{42}$	$+ x(7)W_8^{49}$

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \qquad k = 0, 1, 2, ..., N-1$$

Outline

- Discrete Fourier Transform (DFT)
- Fast Fourier Transform (FFT)
- Twiddle Factor Multiplication
- FFT Algorithms
- FFT Architectures
- Examples
- DIF vs. DIT Decomposition

Fast Fourier Transform (FFT)

- Several fast Fourier transform algorithms have been proposed to reduce the computational complexity of DFT calculation:
  - Prime factor algorithm
  - Winograd algorithm
  - Cooley-Tukey algorithm
    - Most common
    - Focus of this presentation

Fast Fourier Transform (FFT)

- FFT employs the symmetry and periodic properties of the twiddle factors:

$$W_N^{k+N} = W_N^k,$$

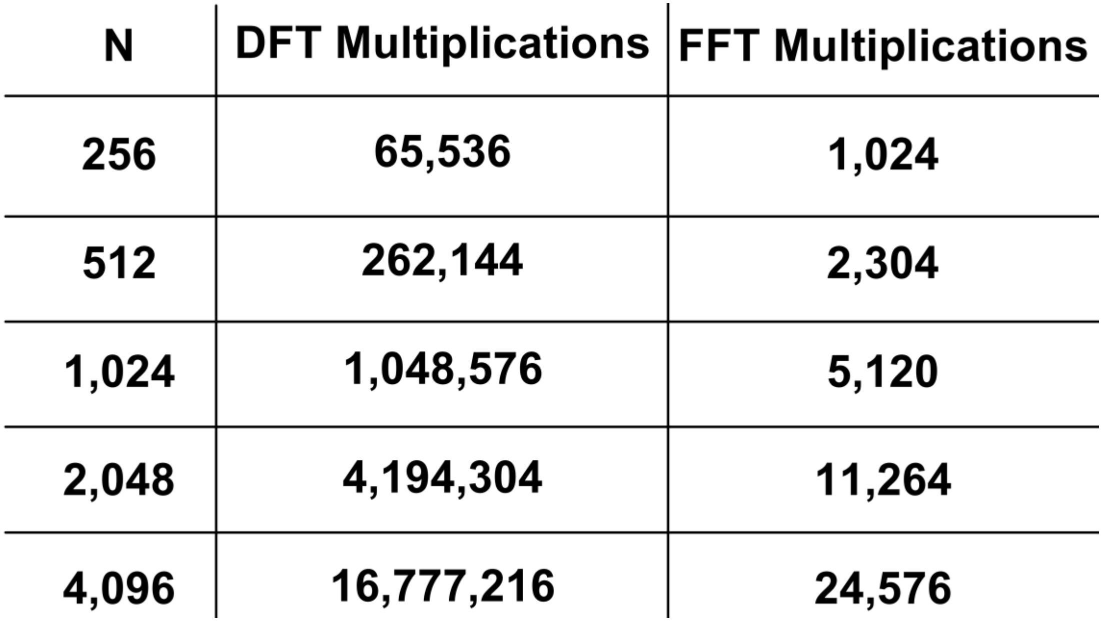
$$W_N^{k+N/2} = -W_N^k$$

, …..

- FFT reduces the computational complexity of DFT calculation to:

$$\mathcal{O}(N * log_2 N)$$

Complexity Reduction



N	DFT Multiplications	FFT Multiplications
256	65,536	1,024
512	262,144	2,304
1,024	1,048,576	5,120
2,048	4,194,304	11,264
4,096	16,777,216	24,576

Outline

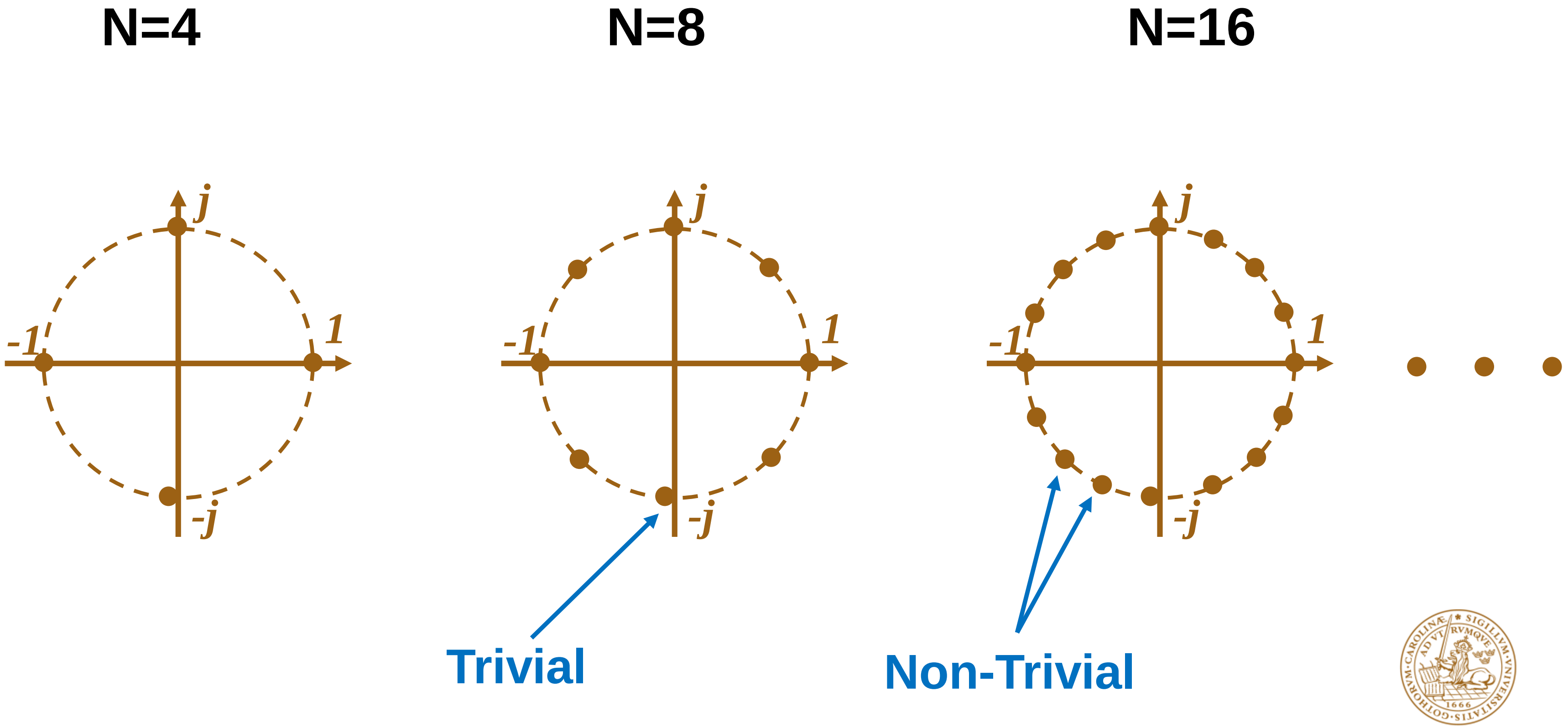
- Discrete Fourier Transform (DFT)
- Fast Fourier Transform (FFT)
- Twiddle Factor Multiplication
- FFT Algorithms
- FFT Architectures
- Data Flow Processing
- DIF vs. DIT Decomposition

LUND UNIVERSITY

Twiddle Factor Multiplication

- There are two types of Twiddle factor multiplications:
  - Trivial
    - Multiplication by ±1, ±j
    - Rotation, …
  - Non-trivial
    - Complex Multiplications

Twiddle Factor Multiplication



N=4

N=8

N=16

Trivial

Non-Trivial

Trivial Rotation

- **Trivial rotation** can be realized by:
  - Interchanging the real and imaginary parts and/or
  - Changing the sign of the real and/or imaginary parts of the input data

Non Trivial Rotation

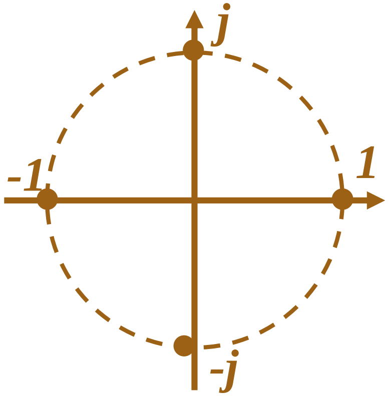
Non trivial rotation can be implemented using:

- General complex multiplier
  - To perform any non-trivial multiplication
- Constant multiplier
  - To perform non-trivial multiplications for specific coefficients
  - Less area
- CORDIC algorithm
  - To realize the non-trivial multiplications through rotation

4-point DFT

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} e^{0} & e^{0} & e^{0} & e^{0} \\ e^{0} & e^{-j2\pi/4} & e^{-j4\pi/4} & e^{-j6\pi/4} \\ e^{0} & e^{-j4\pi/4} & e^{-j8\pi/4} & e^{-j12\pi/4} \\ e^{0} & e^{-j6\pi/4} & e^{-j12\pi/4} & e^{-j18\pi/4} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix}$$

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



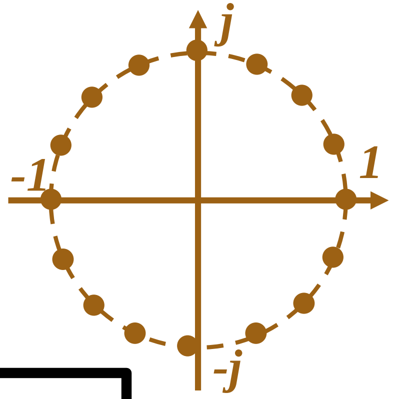
Only trivial coefficients

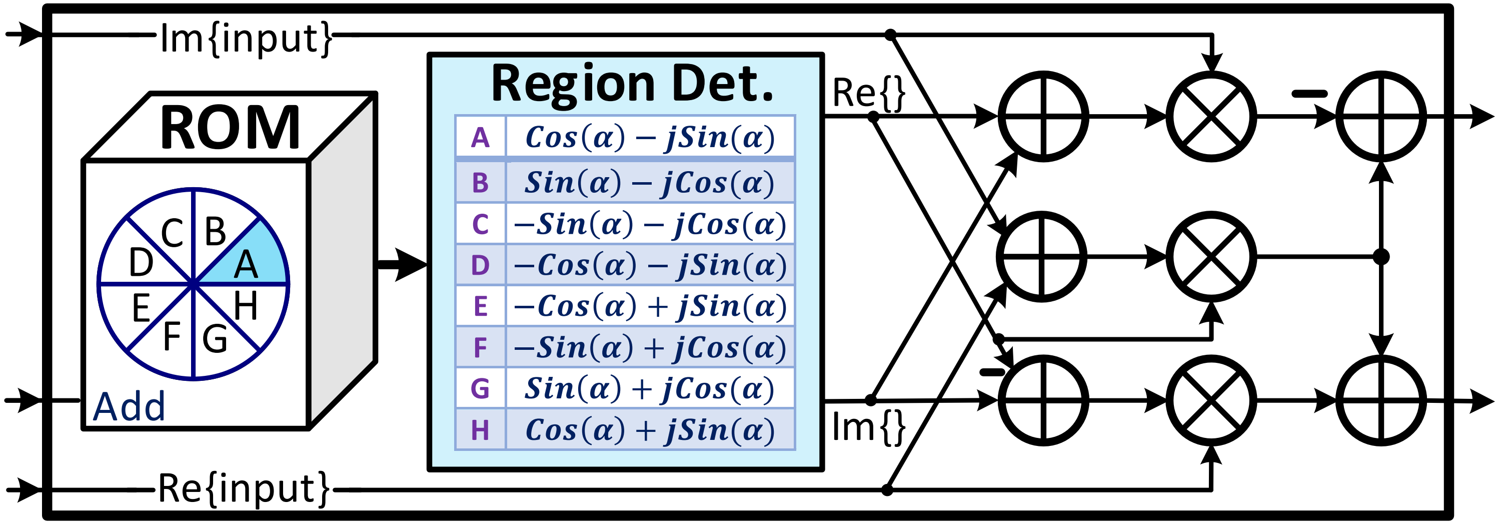
General Twiddle Factor Multiplier

❑ ROM size Reduction:

- ○ Based on the symmetry property, only the coefficients in the first Π/4 region are saved in ROM
- ○ Mapping Table will extract the other coefficients

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



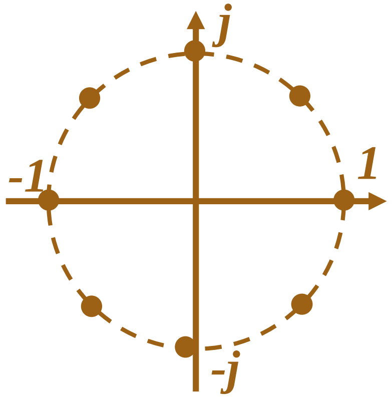


A	$Cos(\alpha)-jSin(\alpha)$
B	$Sin(\alpha)-jCos(\alpha)$
C	$-Sin(\alpha)-jCos(\alpha)$
D	$-Cos(\alpha)-jSin(\alpha)$
E	$-Cos(\alpha)+jSin(\alpha)$
F	$-Sin(\alpha)+jCos(\alpha)$
G	$Sin(\alpha)+jCos(\alpha)$
H	$Cos(\alpha)+jSin(\alpha)$

Concept: Using Coefficient Symmetry

Twiddle Factor Multiplication– Constant Multiplier

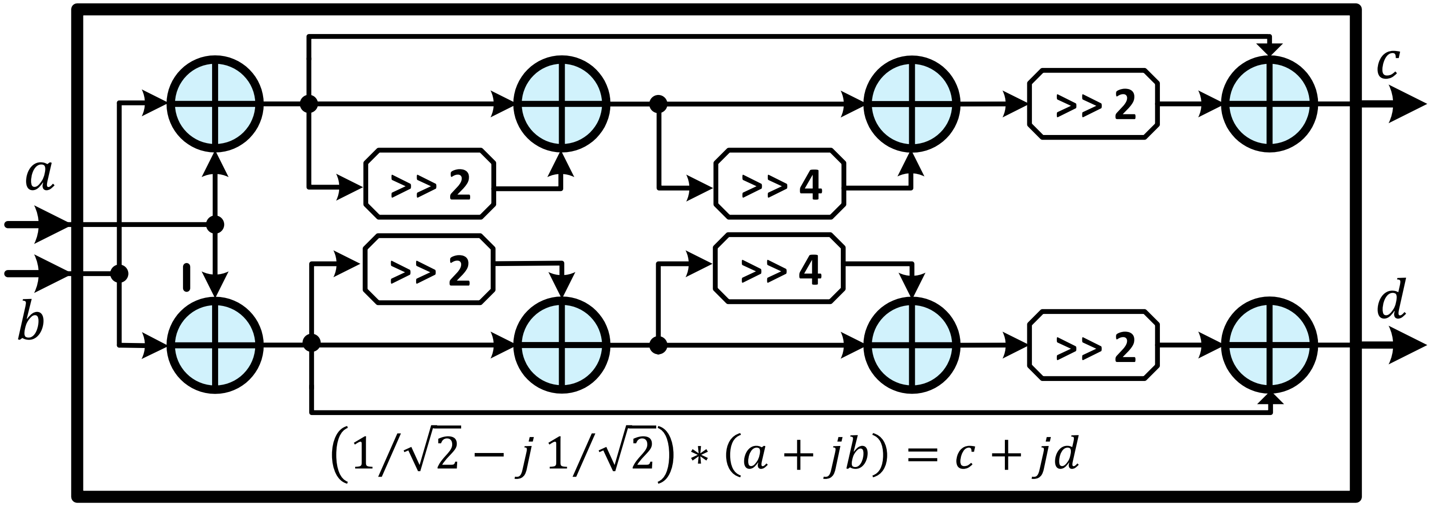
$$W_N^{N/8}(a+jb) = (\frac{1}{\sqrt{2}} - \frac{j}{\sqrt{2}})(a+jb)$$
$$= \frac{1}{\sqrt{2}}[(a+b)+j(b-a)] = c+jd$$



Concept: CSD Representation

$$1/\sqrt{2} = 2^{-1} + 2^{-3} + 2^{-4} + 2^{-6} + 2^{-8}$$

$$1/\sqrt{2} = 1 + (1+2^{-2})(2^{-6} - 2^{-2})$$



$$(1/\sqrt{2} - j\,1/\sqrt{2}) * (a+jb) = c+jd$$

Outline

- Discrete Fourier Transform (DFT)
- Fast Fourier Transform (FFT)
- Twiddle Factor Multiplication
- FFT Algorithms
- FFT Architectures
- Data Flow Processing
- DIF vs. DIT Decomposition

FFT Algorithm

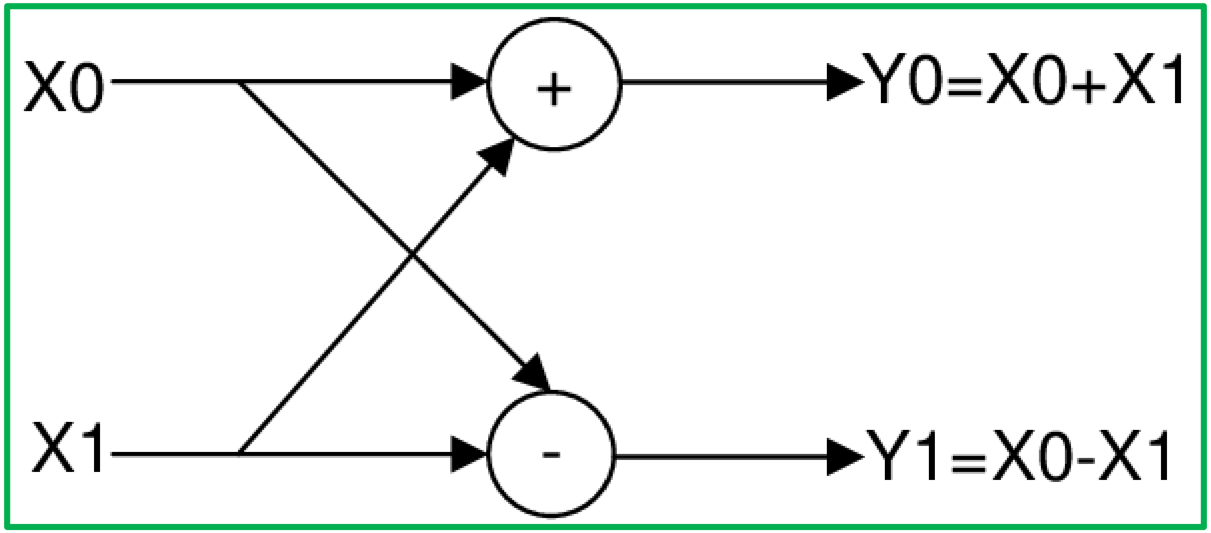
- The most popular FFT algorithms are:
  - Radix-r
  - Improved FFT (Radix-2^n)
  - Mixed-radix
  - Split-radix

Radix-r Algorithm

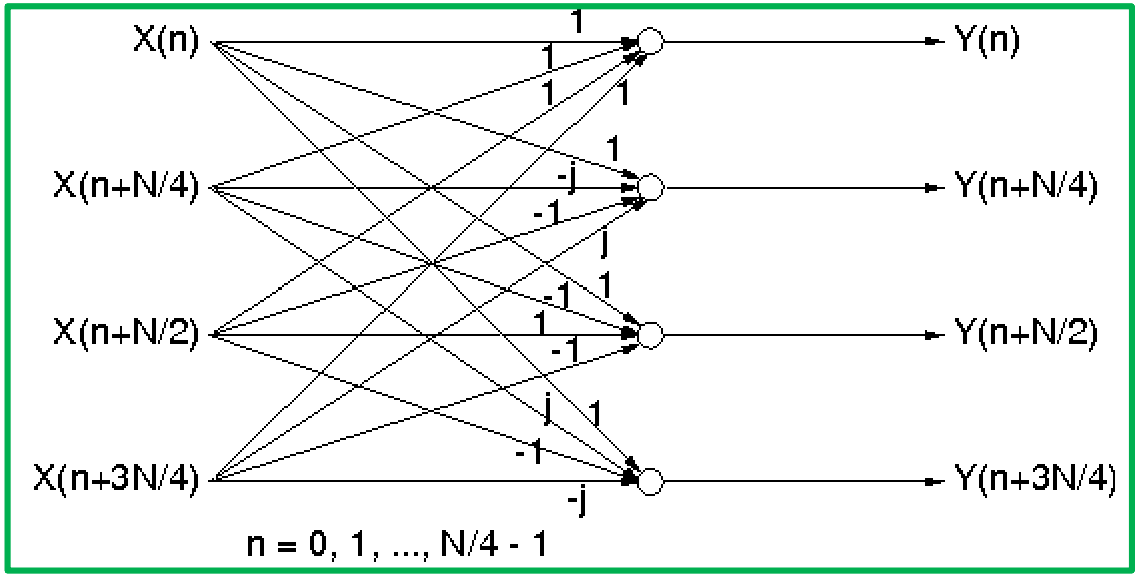
- The radix-r FFT algorithms:
  - DFT of length $\boldsymbol{N}$ is recursively decomposed into $\boldsymbol{N/r}$ and $\boldsymbol{r}$ until all the remaining transform lengths are less than or equal to $\boldsymbol{r}$.
  - Number of stages: $\mathbf{log}_{\boldsymbol{r}} \boldsymbol{N}$
  - A **high radix** FFT algorithm reduces the number of processing stages
    - Increases the hardware complexity of each stage significantly.

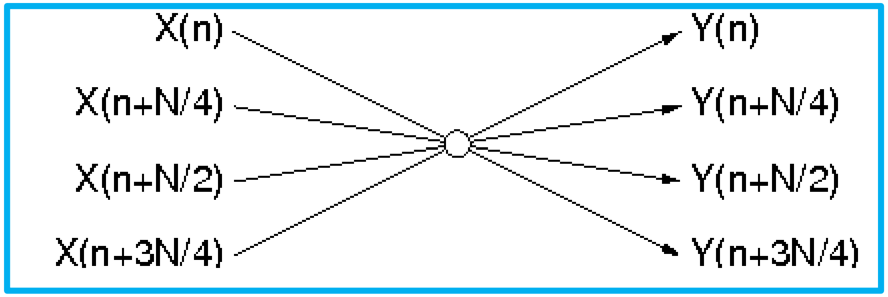
Radix-2 Butterfly

- Complex inputs/outputs

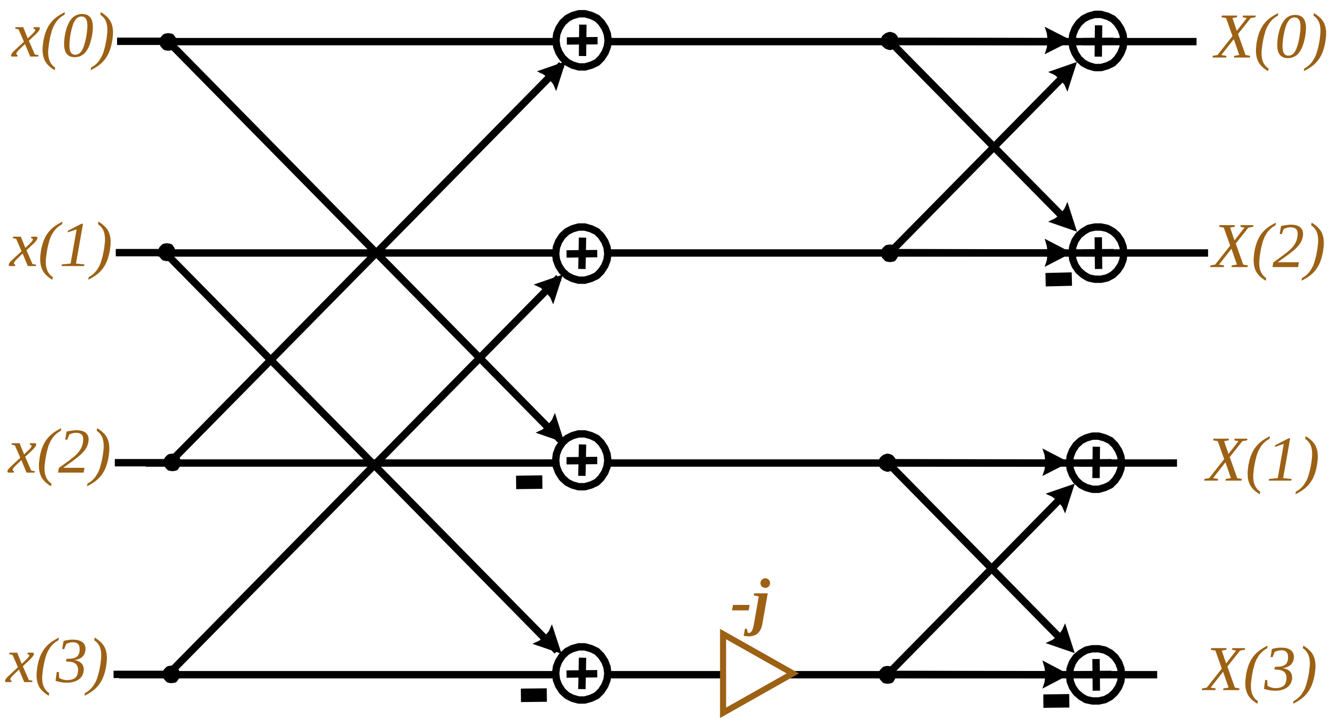


Radix-4 Butterfly

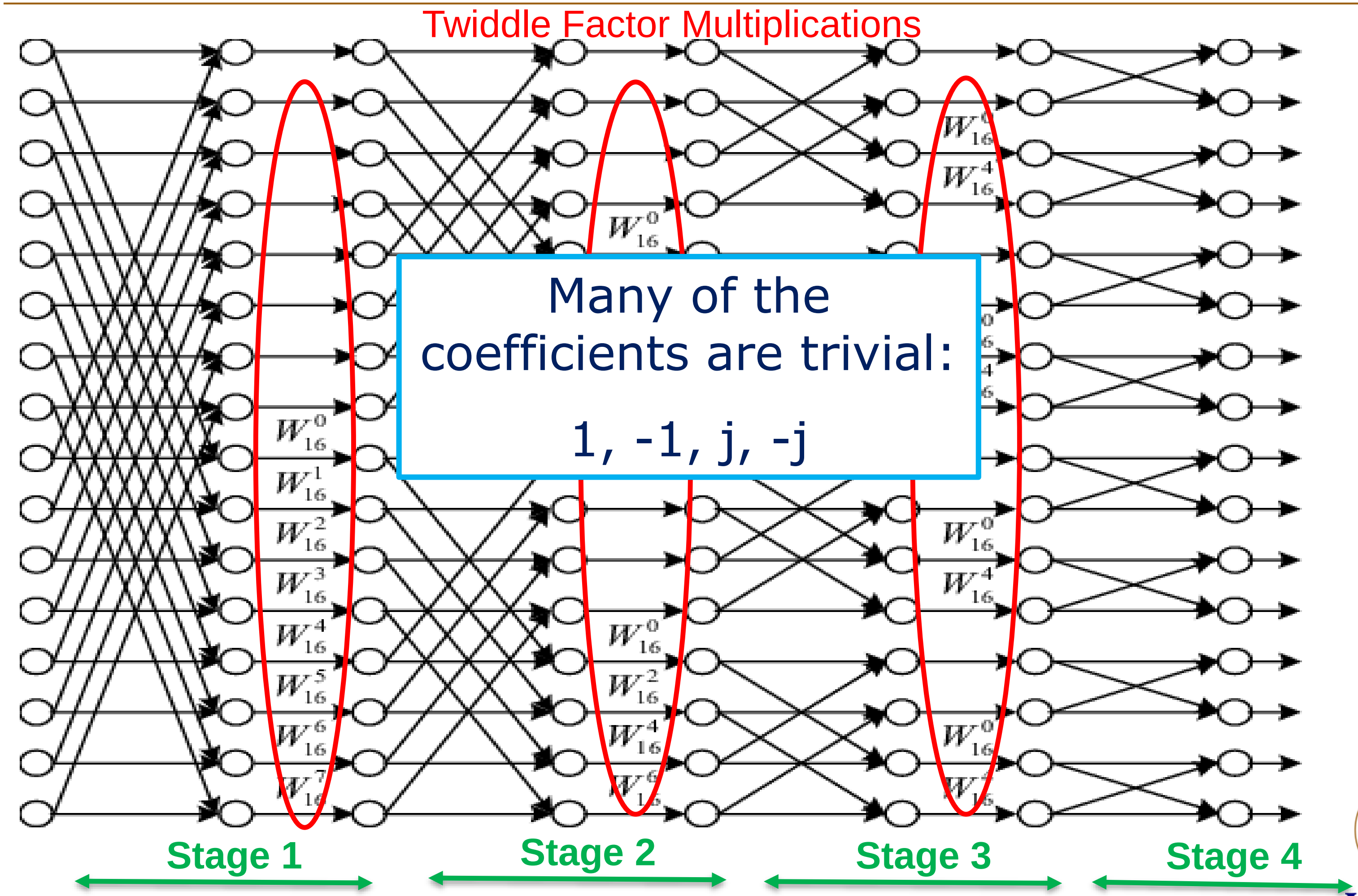




4-point FFT with Radix-2 Butterfly



16-point FFT - Radix-2 Algorithm

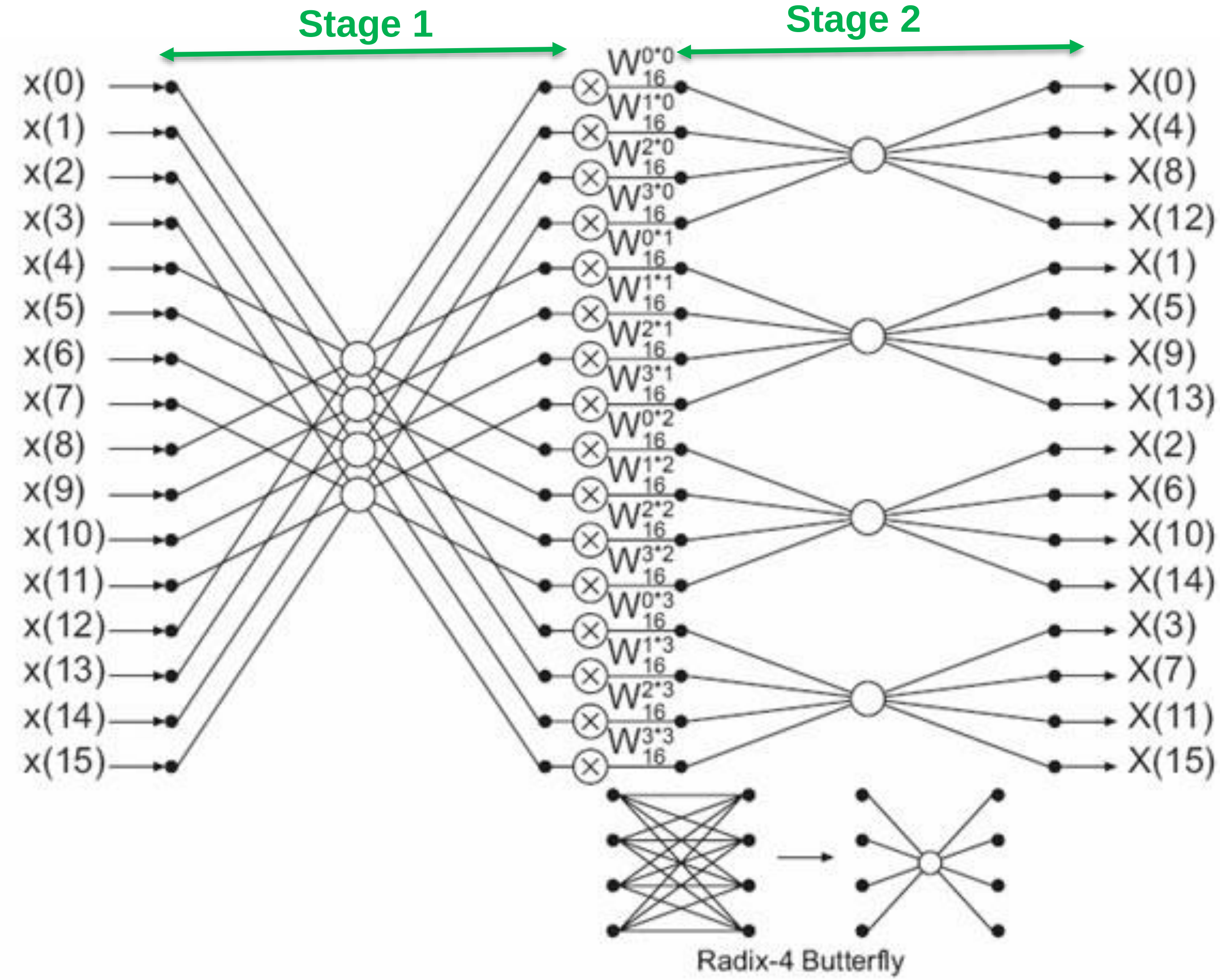


Many of the coefficients are trivial:

1, -1, j, -j

Number of stages: $\log_2(N)$

16-point Radix-4 FFT



Small-radix vs. High-radix FFT Algorithm

- Selection of radix has a large impact on the complexity of FFT algorithm
- **Small radix** FFT architecture**:**
  - Simple butterfly operation
  - Higher number of twiddle factor multiplications
- High-radix pipelined FFT architectures have been proposed to improve the arithmetic resource utilization.

Small-radix vs. High-radix FFT Algorithm

- **High-radix** FFT:
  - The more efficient use of multipliers and adders
  - Less number of twiddle factor multiplications
  - Reduces the number of stages
  - More complexity in trivial twiddle factor computation
  - More complex stage (i.e. butterfly units)
    - Radices higher than 4 require butterflies with non trivial rotations.

Improved FFT (Radix-2^n) Algorithm

- Radix- 2^n algorithms are proposed to overcome the drawback of high-radix algorithms.

- Radix-2^n algorithm can be explained by applying the CT algorithm two times.
  - Basic unit of decomposition consists of the radix-2 butterfly.
  - The number of stages requiring twiddle factor multiplications is reduced.

e.g. Radix-2^2 Algorithm

- This algorithm has:
  - The same number of non-trivial multiplications as a radix-4 algorithm
  - The same butterfly structure as that of radix-2 algorithm
    - Can be mapped to radix-2 butterflies.

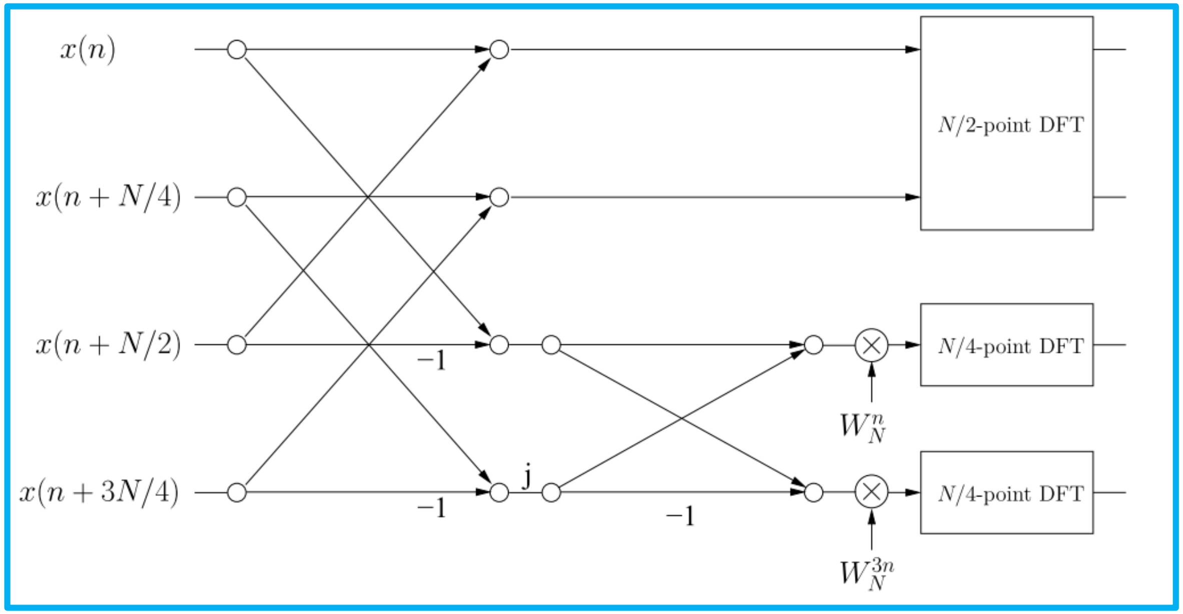
- This can further on extended to Radix-2^3 and 2^4.

Mixed-radix Algorithm

- The mixed-radix algorithms can be derived by mixing different radixes.
  - ○ Generate desired FFT lengths
  - ○ More efficient processing
  - ○ Hardware complexity is similar to radix-2^n algorithm

Split-radix Algorithm

- The main idea is that independent parts of the algorithm should be computed independently based on the best possible computational scheme.
  - Reduction in computational complexity



Split-radix Algorithm

- In split-radix algorithms for 2^n size DFT:
  - The total number of complex multiplications can be reduced
  - Each stage becomes irregular
  - Not efficient in terms of pipelined processing
  - More complex control due to the irregularity

Outline

- Discrete Fourier Transform (DFT)
- Fast Fourier Transform (FFT)
- Twiddle Factor Multiplication
- FFT Algorithms
- FFT Architectures
- Data Flow Processing
- DIF vs. DIT Decomposition

FFT Architectures

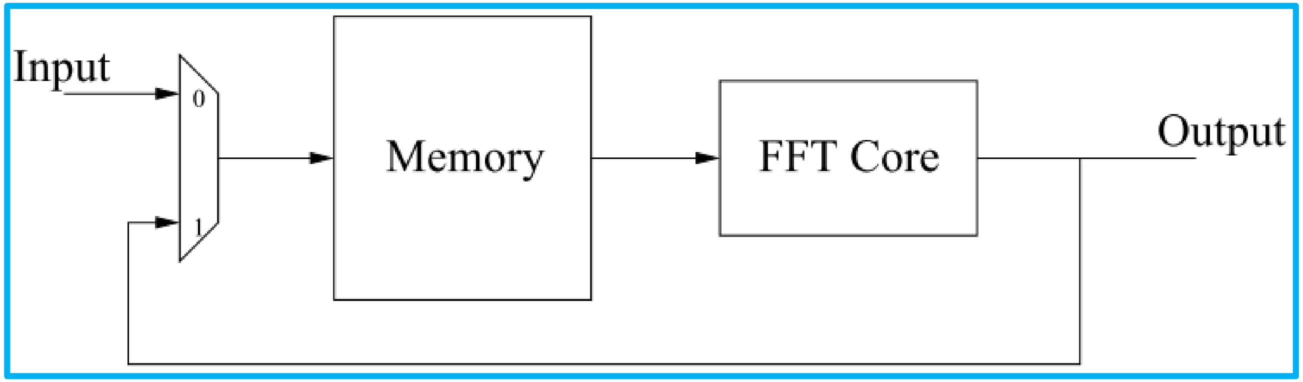
- Most FFT architectures can be categorized into:
  - Direct implementation
  - Memory-based
  - Pipelined

Direct Implementation

- Requires a number of processing elements equal to the number of operations
  - Very hardware intensive
  - It can be suitable for small size FFTs
  - The utilization of the butterflies and rotators is 100%

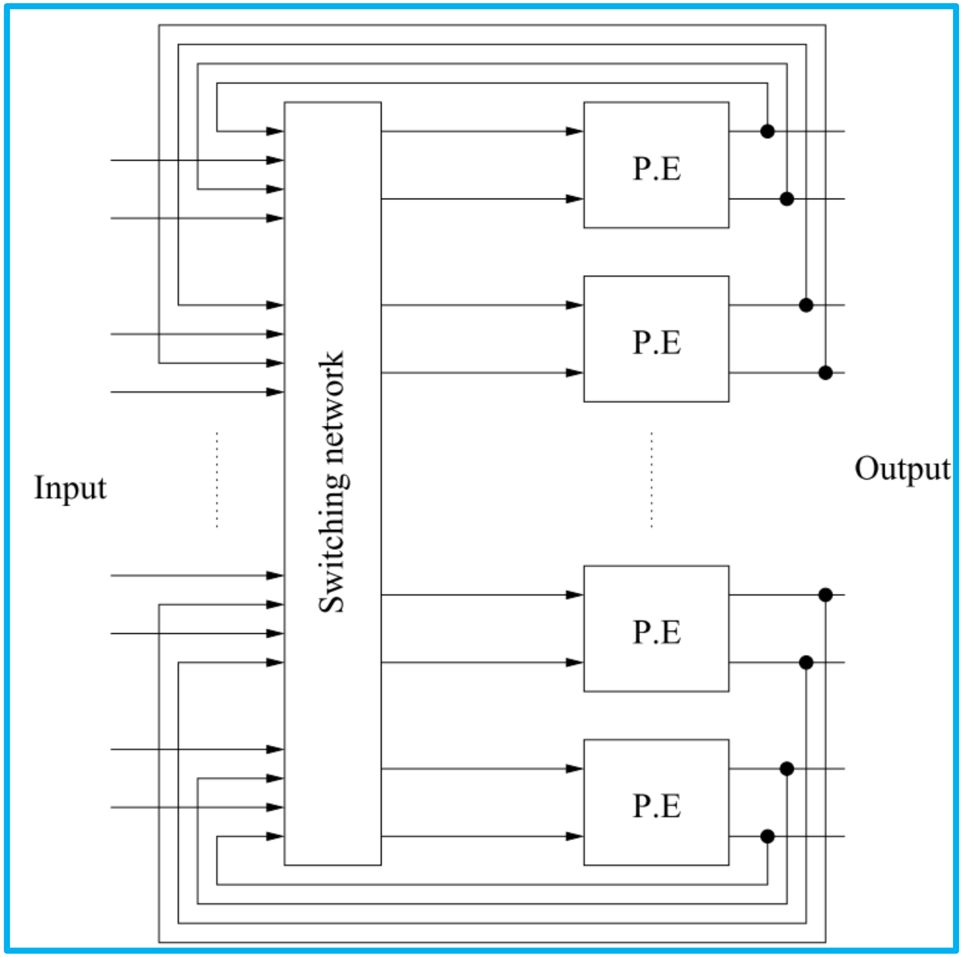
Memory Based Architectures

- One or more processing elements (Pes) calculate all the butterflies and twiddle factor multiplications.
  - It is necessary to compute whole FFT before it receives new samples.
  - Unable to compute the FFT when data arrives continuously.
    - This can be solved by adding extra memory



Concept: Folding & Time Multiplexing

Memory Based Architectures



Concept: Unfolding/Parallel Processing

Memory Based Architectures

- Memory-based architectures (in-place architecture):
  - Smaller area
  - Low power
  - Long latency
  - Require additional buffer space
  - Lower throughput compared to the pipelined architectures
    - Parallel processing is used to improve throughput and latency.
      - Hardware cost is increased
    - High-radix processing elements are used to improve throughput.
      - It causes memory conflict problems

Not suitable for FFT computation in real time applications

Pipelined Architectures

- Two principal techniques for pipelined architectures:
  - Delay Feedback (DF), often referred to as Feedback
    - SDF
    - MDF
  - Delay Commutator (DC), often referred to as Feed Forward (FF)
    - MDC
    - SDC

Pipelined architecture is a proper choice for **high-throughput** and **real time** applications

Single-path Delay Feedback Architectures

- **SDF-based** architectures provide memory feedback paths to manage some butterfly outputs during each stage.

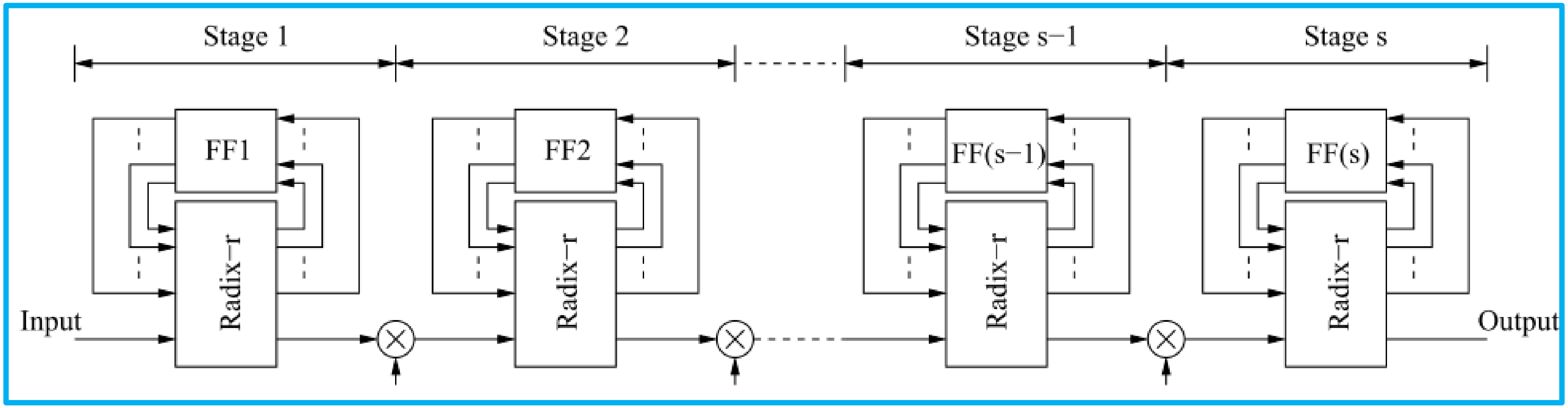
- **SDF** techniques allow the initial FFT output sample to be generated instantly after the final FFT input sample has been processed.

- **SDF** architecture has one continuous data stream of one sample per clock cycle

Pipeline Architectures

- The pipelined FFT architectures:
    - Higher throughput
    - Lower latency
    - Suitable for real-time applications
    - Acceptable hardware cost
    - Perform non-stop processing at sample rate
    - Proper for low power solution

Single-path Delay Feedback Architectures



Single-path Delay Feedback Architectures

- **SDF** architecture has:
  - **Lower Latency!**
  - Low cost
  - High hardware efficiency
  - Low throughput due to the single path
    - No concurrent processing
  - Arithmetic utilization is relatively low (50%)

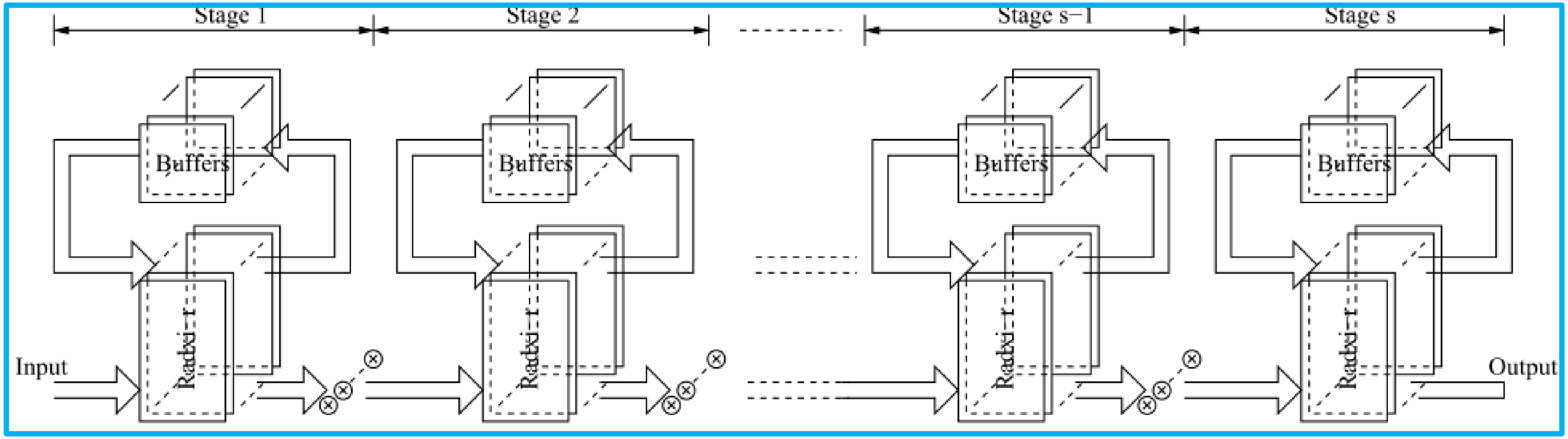
SDF is an optimal choice in terms of the hardware cost and performance for many applications

Multipath Delay Feedback Architectures

- **MDF** architecture can be generated by extending the **SDF** FFT architecture using a multiple-path approach.
  - A solution to provide a higher throughput
  - Higher hardware cost
  - Arithmetic utilization is relatively low (50%)

- Multiple-path (M) architectures, are often adopted for high throughput applications

Multipath Delay Feedback Architectures



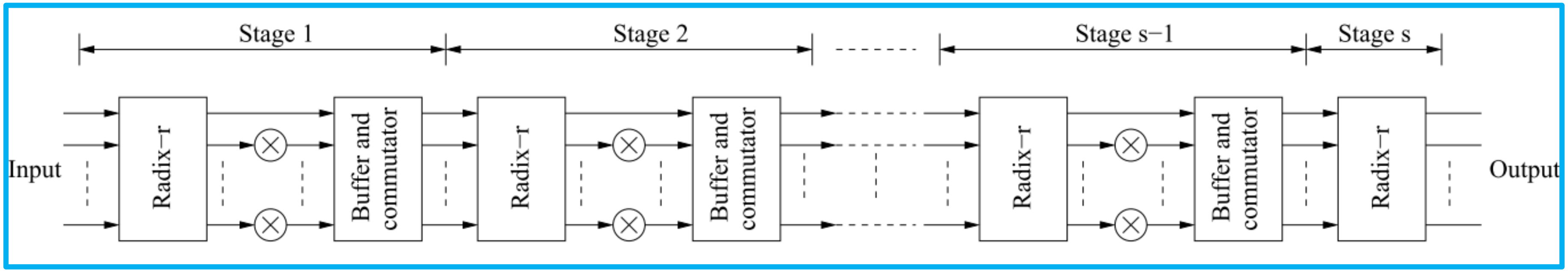
Concept: Unfolding/Parallel Processing

Multi Delay Commutator Architectures

- **MDC-based** architectures replace feedback data paths with feed forward data paths with commutators as switching operations.
  - Each stage forwards its output to the next without any feedback
  - **MDC** architecture processes several samples in parallel

- These architectures can be improved by using radix-2^n.

Multi Delay Commutator Architectures

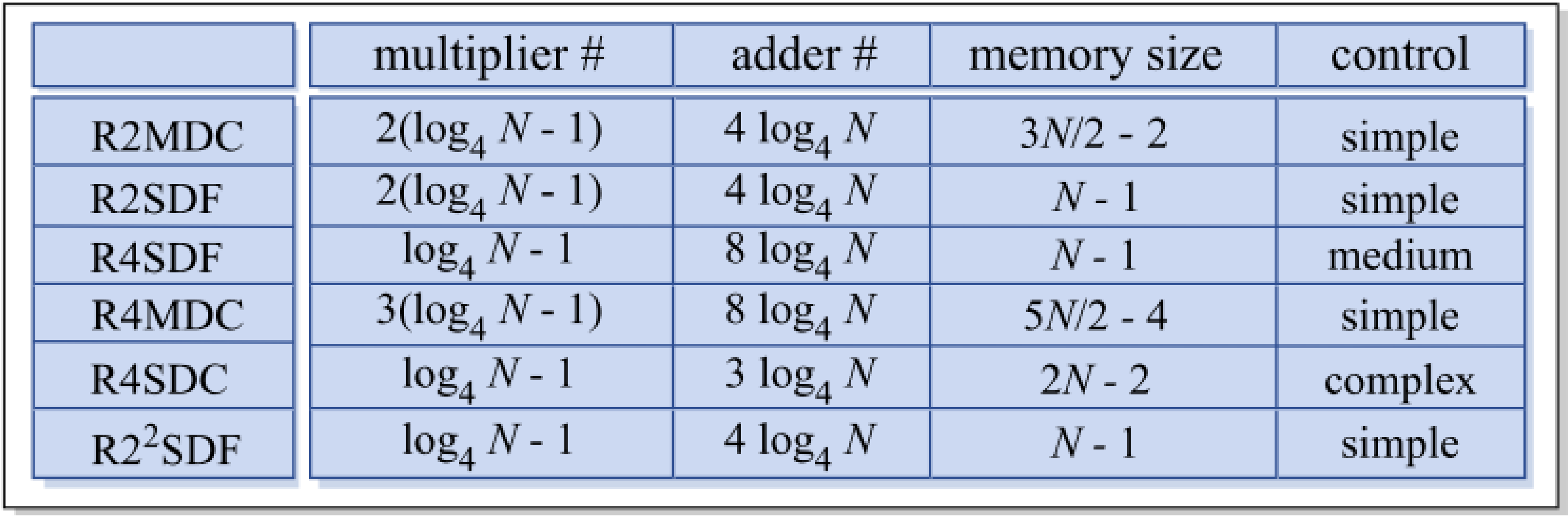


Multi Delay Commutator Architectures

- **MDC-based** architecture:
  - Simple control path
  - 100% utilization ratio of butterflies
  - Higher throughput than **SDF**
  - Higher hardware cost

MDC can achieve higher throughput, while **SDF** needs less memory and hardware cost.

Algorithm/Architecture Comparison



	multiplier #	adder #	memory size	control
R2MDC	$2(\log_4 N - 1)$	$4 \log_4 N$	$3N/2 - 2$	simple
R2SDF	$2(\log_4 N - 1)$	$4 \log_4 N$	$N - 1$	simple
R4SDF	$\log_4 N - 1$	$8 \log_4 N$	$N - 1$	medium
R4MDC	$3(\log_4 N - 1)$	$8 \log_4 N$	$5N/2 - 4$	simple
R4SDC	$\log_4 N - 1$	$3 \log_4 N$	$2N - 2$	complex
R2^2SDF	$\log_4 N - 1$	$4 \log_4 N$	$N - 1$	simple

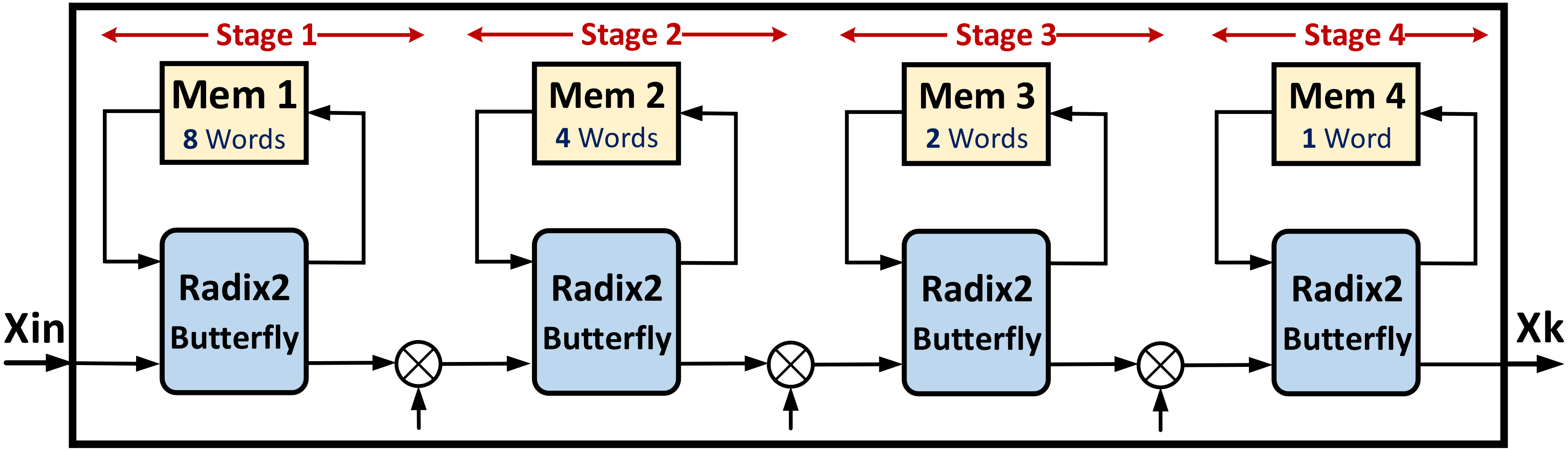
Outline

- Discrete Fourier Transform (DFT)
- Fast Fourier Transform (FFT)
- Twiddle Factor Multiplication
- FFT Algorithms
- FFT Architectures
- Data Flow Processing
- DIF vs. DIT Decomposition

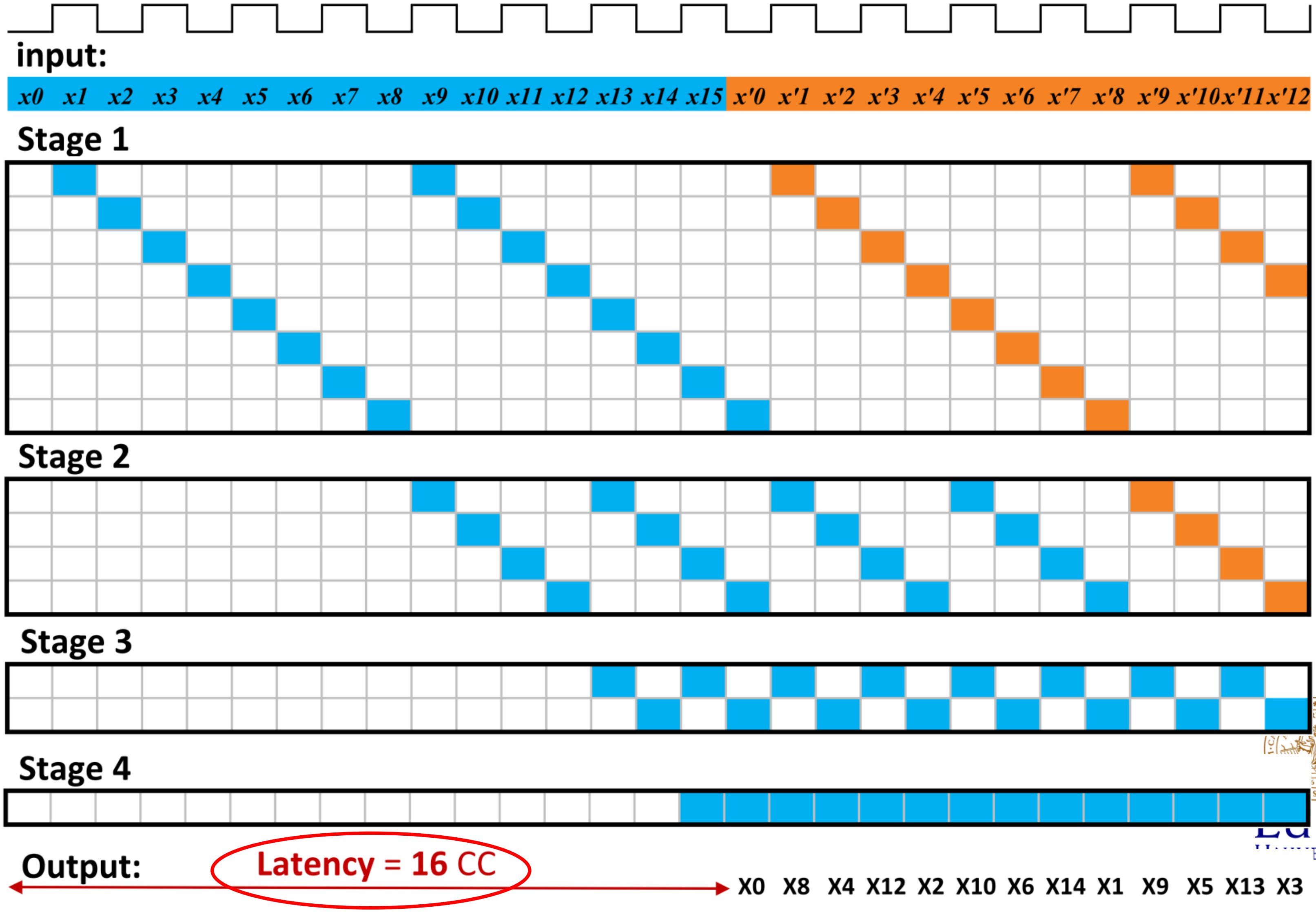
LUND
UNIVERSITY

SDF Architecture

- 16-point Single-input Pipelined FFT



16-point SDF Architexture



input:

x0 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x'0 x'1 x'2 x'3 x'4 x'5 x'6 x'7 x'8 x'9 x'10 x'11 x'12

Stage 1

Stage 2

Stage 3

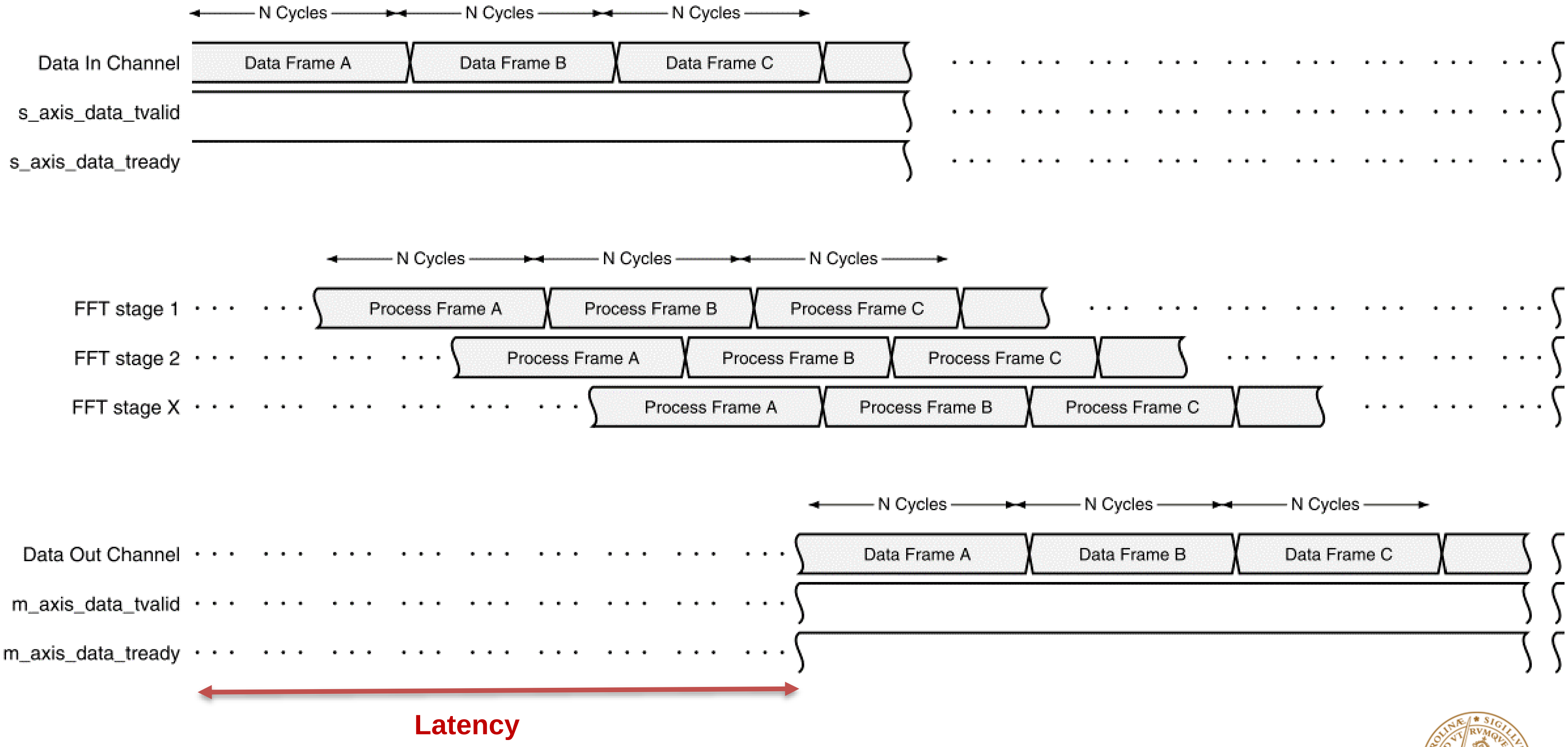
Stage 4

Output:

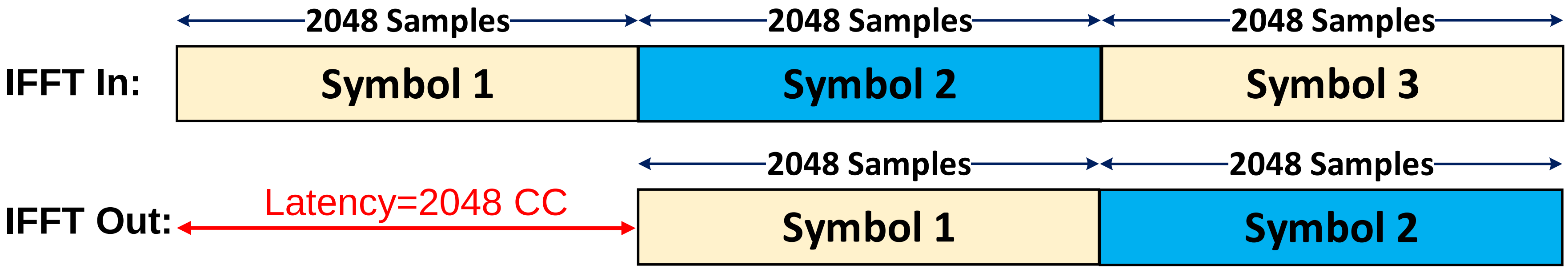
Latency = 16 CC

X0 X8 X4 X12 X2 X10 X6 X14 X1 X9 X5 X13 X3

Timing Diagram of Pipelined FFT



2048-point SDF Architecture



IFFT

- IFFT is realized as:

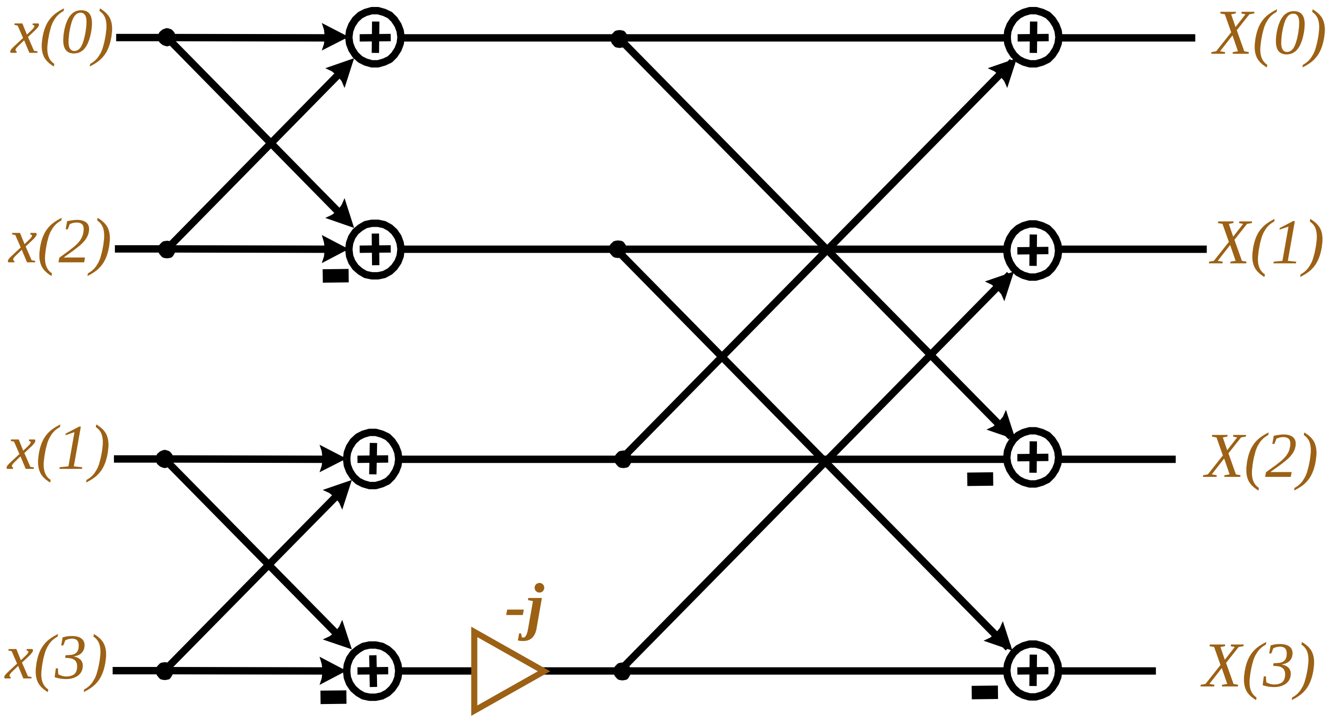
$$x(n) = \frac{1}{N}\left(\sum_{k=0}^{N-1} X(k)^* W_N^{nk}\right)^*, \qquad n = 0, 1, ..., N-1$$

- The same hardware can be used

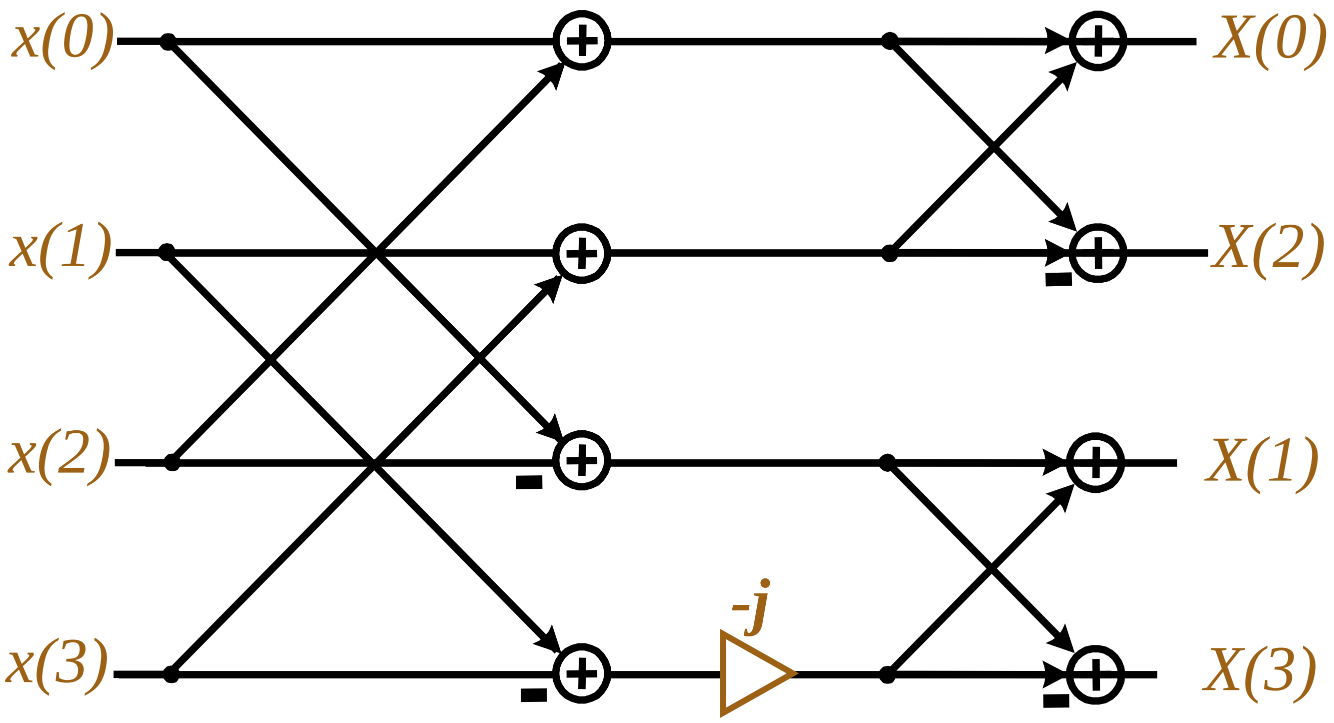
Outline

- Discrete Fourier Transform (DFT)
- Fast Fourier Transform (FFT)
- Twiddle Factor Multiplication
- FFT Algorithms
- FFT Architectures
- Data Flow Processing
- DIF vs. DIT Decomposition

4-point FFT with Radix-2 Butterfly



4-point FFT with Radix-2 Butterfly



DIF vs. DIT Decomposition

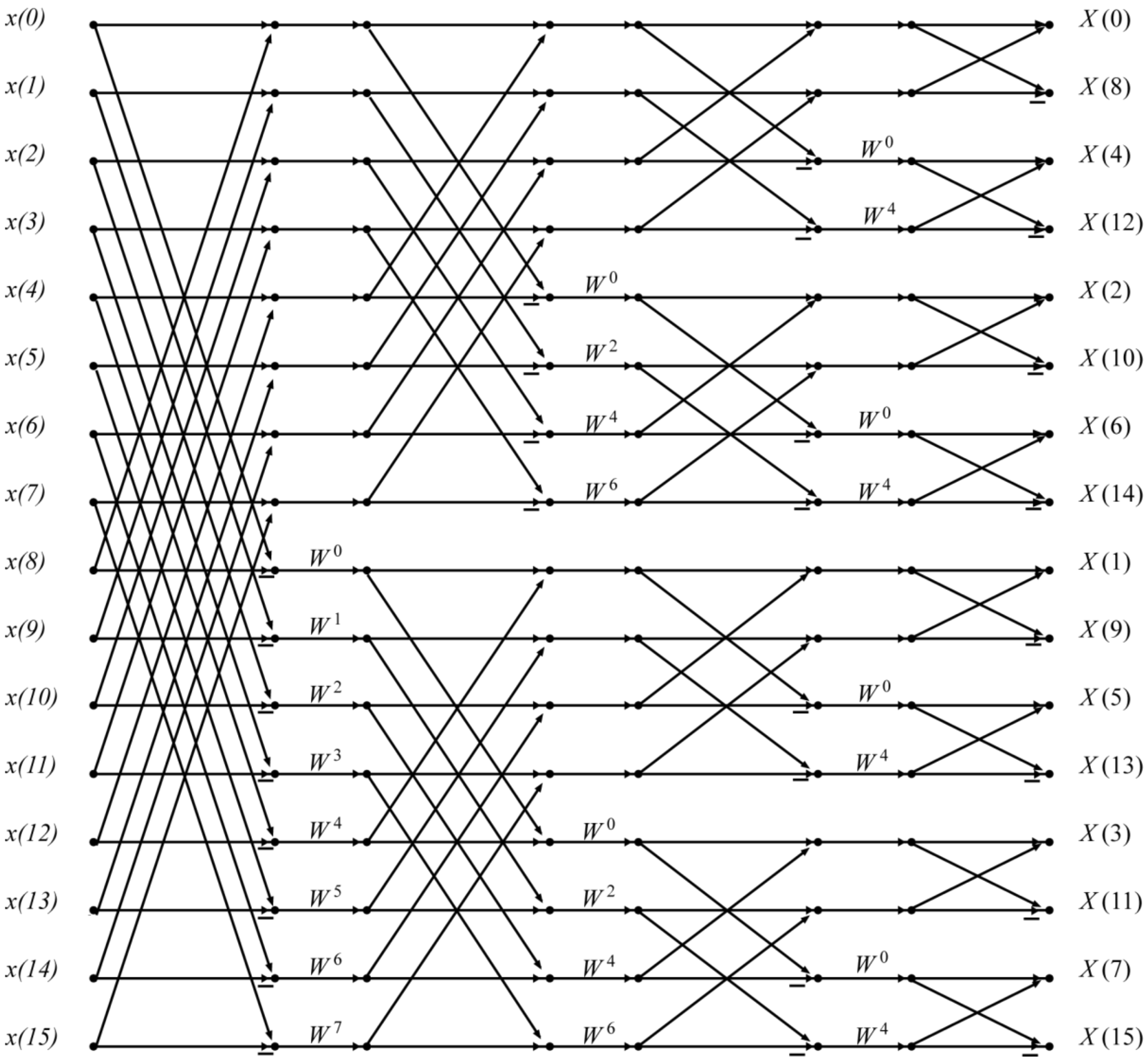
- According to the decomposition direction, FFT algorithms can be classified into:
  - **DIF** decomposition:
    - The output sequence is separated into even and odd indexed samples iteratively.
  - **DIT** decomposition:
    - Separates the input sequence into even and odd samples iteratively.

DIF vs. DIT Decomposition

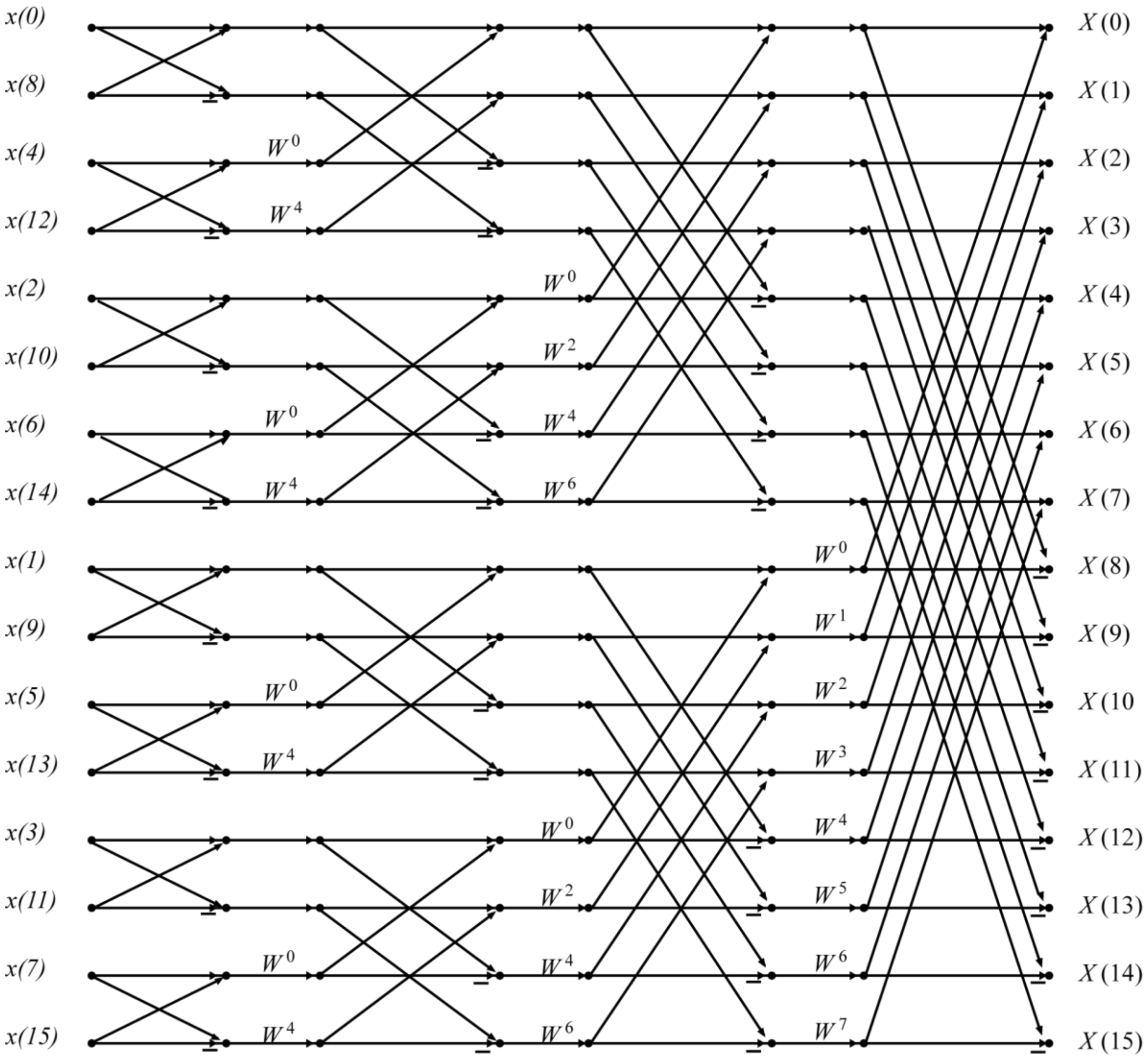
- In **DIF**, the input samples are usually in order and the output samples are in bit-reversed order.
- In **DIT**, the input samples are usually in bit-reversed order and the output samples are in natural order.

- The location of the twiddle factor multiplications
- Input/Output Order

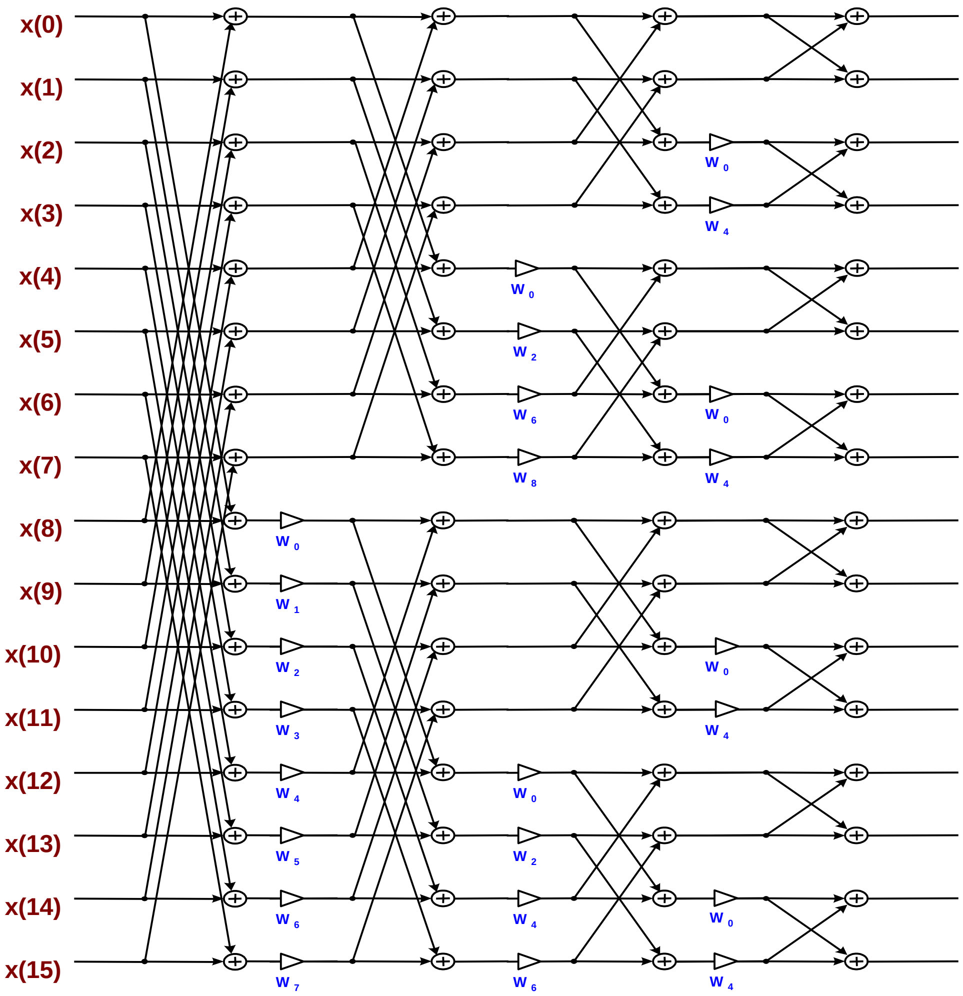
16-point DIF


x(0)
x(1)
x(2)
x(3)
x(4)
x(5)
x(6)
x(7)
x(8)
x(9)
x(10)
x(11)
x(12)
x(13)
x(14)
x(15)
W 0
W 1
W 2
W 3
W 4
W 5
W 6
W 7
W 0
W 2
W 4
W 6
W 0
W 2
W 4
W 6
W 0
W 4
W 0
W 4
W 0
W 4
W 0
W 4
X (0)
X (8)
X (4)
X (12)
X (2)
X (10)
X (6)
X (14)
X (1)
X (9)
X (5)
X (13)
X (3)
X (11)
X (7)
X (15)

16-point DIT


x(0)
x(8)
x(4)
x(12)
x(2)
x(10)
x(6)
x(14)
x(1)
x(9)
x(5)
x(13)
x(3)
x(11)
x(7)
x(15)
W^0
W^4
W^0
W^4
W^0
W^4
W^0
W^4
W^0
W^2
W^4
W^6
W^0
W^2
W^4
W^6
W^0
W^1
W^2
W^3
W^4
W^5
W^6
W^7
X(0)
X(1)
X(2)
X(3)
X(4)
X(5)
X(6)
X(7)
X(8)
X(9)
X(10
X(11)
X(12)
X(13)
X(14)
X(15)

16-Point FFT



Bit-reversed		"Normal"
X(0)		
X(8) -	1000	X(1) - 0001
X(4) -	0100	X(2) - 0010
X(12) -	1100	X(3) - 0011
X(2) -	0010	X(4) - 0100
X(10) -	1010	X(5) - 0101
X(6) -	0110	X(6) - 0110
X(14) -	1110	X(7) - 0111
X(1) -	0001	X(8) - 1000
X(9) -	1001	X(9) - 1001
X(5) -	0101	X(10) - 1010
X(13) -	1101	X(11) - 1011
X(3) -	0011	X(12) - 1100
X(11) -	1011	X(13) - 1101
X(7) -	0111	X(14) - 1110
X(15) -	1111	X(15) - 1111

A Reordering Circuit is needed to perform the above conversion

LUND
UNIVERSITY