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:

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

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

<table>
<thead>
<tr>
<th>$X(0)$</th>
<th>$x(0)W_8^0 + x(1)W_8^0 + x(2)W_8^0 + x(3)W_8^0 + x(4)W_8^0 + x(5)W_8^0 + x(6)W_8^0 + x(7)W_8^0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X(1)$</td>
<td>$x(0)W_8^0 + x(1)W_8^1 + x(2)W_8^2 + x(3)W_8^3 + x(4)W_8^4 + x(5)W_8^5 + x(6)W_8^6 + x(7)W_8^7$</td>
</tr>
<tr>
<td>$X(2)$</td>
<td>$x(0)W_8^0 + x(1)W_8^2 + x(2)W_8^4 + x(3)W_8^6 + x(4)W_8^8 + x(5)W_8^{10} + x(6)W_8^{12} + x(7)W_8^{14}$</td>
</tr>
<tr>
<td>$X(3)$</td>
<td>$x(0)W_8^0 + x(1)W_8^3 + x(2)W_8^6 + x(3)W_8^9 + x(4)W_8^{12} + x(5)W_8^{15} + x(6)W_8^{18} + x(7)W_8^{21}$</td>
</tr>
<tr>
<td>$X(4)$</td>
<td>$x(0)W_8^0 + x(1)W_8^4 + x(2)W_8^8 + x(3)W_8^{12} + x(4)W_8^{16} + x(5)W_8^{20} + x(6)W_8^{24} + x(7)W_8^{28}$</td>
</tr>
<tr>
<td>$X(5)$</td>
<td>$x(0)W_8^0 + x(1)W_8^5 + x(2)W_8^{10} + x(3)W_8^{15} + x(4)W_8^{20} + x(5)W_8^{25} + x(6)W_8^{30} + x(7)W_8^{35}$</td>
</tr>
<tr>
<td>$X(6)$</td>
<td>$x(0)W_8^0 + x(1)W_8^6 + x(2)W_8^{12} + x(3)W_8^{18} + x(4)W_8^{24} + x(5)W_8^{30} + x(6)W_8^{36} + x(7)W_8^{42}$</td>
</tr>
<tr>
<td>$X(7)$</td>
<td>$x(0)W_8^0 + x(1)W_8^7 + x(2)W_8^{14} + x(3)W_8^{21} + x(4)W_8^{28} + x(5)W_8^{35} + x(6)W_8^{42} + x(7)W_8^{49}$</td>
</tr>
</tbody>
</table>

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

\[ \mathcal{O}(N \times \log_{2}N) \]
# Complexity Reduction

<table>
<thead>
<tr>
<th>$N$</th>
<th>DFT Multiplications</th>
<th>FFT Multiplications</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>65,536</td>
<td>1,024</td>
</tr>
<tr>
<td>512</td>
<td>262,144</td>
<td>2,304</td>
</tr>
<tr>
<td>1,024</td>
<td>1,048,576</td>
<td>5,120</td>
</tr>
<tr>
<td>2,048</td>
<td>4,194,304</td>
<td>11,264</td>
</tr>
<tr>
<td>4,096</td>
<td>16,777,216</td>
<td>24,576</td>
</tr>
</tbody>
</table>
Outline

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

There are two types of Twiddle factor multiplications:

- **Trivial**
  - Multiplication by $\pm 1, \pm j$
  - 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 $\pi/4$ region are saved in ROM.
  - Mapping Table will extract the other coefficients.

\[
X(k) = \sum_{n=0}^{N-1} x(n)W_n^k, \quad k = 0, 1, \ldots, N - 1
\]

Concept: Using Coefficient Symmetry
Twiddle Factor Multiplication–Constant Multiplier

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

**Concept: CSD Representation**

\[ \frac{1}{\sqrt{2}} = 2^{-1} + 2^{-3} + 2^{-4} + 2^{-6} + 2^{-8} \]
\[ \frac{1}{\sqrt{2}} = 1 + (1 + 2^{-2})(2^{-6} - 2^{-2}) \]

\[ (1/\sqrt{2} - j 1/\sqrt{2}) \times (a + jb) = c + jd \]
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 $N$ is recursively decomposed into $N/r$ and $r$ until all the remaining transform lengths are less than or equal to $r$.

- Number of stages: $\log_r N$

- 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

\[ Y_0 = X_0 + X_1 \]

\[ Y_1 = X_0 - X_1 \]
Radix-4 Butterfly

\[ n = 0, 1, \ldots, N/4 - 1 \]
4-point FFT with Radix-2 Butterfly

\[ x(0) \rightarrow x(1) \rightarrow x(2) \rightarrow x(3) \]

\[ X(0) \rightarrow X(2) \rightarrow X(1) \rightarrow X(3) \]
16-point FFT - Radix-2 Algorithm

Twiddle Factor Multiplications

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
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

<table>
<thead>
<tr>
<th></th>
<th>multiplier #</th>
<th>adder #</th>
<th>memory size</th>
<th>control</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2MDC</td>
<td>2(log(_4) N - 1)</td>
<td>4 log(_4) N</td>
<td>(3N/2 - 2)</td>
<td>simple</td>
</tr>
<tr>
<td>R2SDF</td>
<td>2(log(_4) N - 1)</td>
<td>4 log(_4) N</td>
<td>(N - 1)</td>
<td>simple</td>
</tr>
<tr>
<td>R4SDF</td>
<td>log(_4) N - 1</td>
<td>8 log(_4) N</td>
<td>(N - 1)</td>
<td>medium</td>
</tr>
<tr>
<td>R4MDC</td>
<td>3(log(_4) N - 1)</td>
<td>8 log(_4) N</td>
<td>(5N/2 - 4)</td>
<td>simple</td>
</tr>
<tr>
<td>R4SDC</td>
<td>log(_4) N - 1</td>
<td>3 log(_4) N</td>
<td>(2N - 2)</td>
<td>complex</td>
</tr>
<tr>
<td>R(2)(^2)SDF</td>
<td>log(_4) N - 1</td>
<td>4 log(_4) N</td>
<td>(N - 1)</td>
<td>simple</td>
</tr>
</tbody>
</table>
Outline

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

16-point Single-input Pipelined FFT
16-point SDF Architecture

input:

Stage 1

Stage 2

Stage 3

Stage 4

Output: Latency = 16 CC
Timing Diagram of Pipelined FFT

Data In Channel
- Data Frame A
- Data Frame B
- Data Frame C

s_axis_data_tvalid
s_axis_data_tready

FFT stage 1
- Process Frame A
- Process Frame B
- Process Frame C

FFT stage 2
- Process Frame A
- Process Frame B
- Process Frame C

FFT stage X
- Process Frame A
- Process Frame B
- Process Frame C

Data Out Channel
- Data Frame A
- Data Frame B
- Data Frame C

m_axis_data_tvalid
m_axis_data_tready

Latency
2048-point SDF Architecture

IFFT In: 2048 Samples  
Symbol 1  Symbol 2  Symbol 3

IFFT Out: 2048 Samples
Symbol 1  Symbol 2

Latency = 2048 CC
IFFT

IFFT is realized as:

\[ x(n) = \frac{1}{N} \left( \sum_{k=0}^{N-1} X(k) W_n^{-nk} \right)^*, \quad n = 0, 1, ..., N - 1 \]

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

\[ x(0) \rightarrow x(2) \rightarrow x(1) \rightarrow x(3) \rightarrow X(0) \rightarrow X(1) \rightarrow X(2) \rightarrow X(3) \]
4-point FFT with Radix-2 Butterfly

\[ x(0) \rightarrow x(1) \rightarrow x(2) \rightarrow x(3) \rightarrow X(0) \rightarrow X(2) \rightarrow X(1) \rightarrow X(3) \]
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

\[ x(0) \rightarrow X(0) \]
\[ x(1) \rightarrow X(8) \]
\[ x(2) \rightarrow X(4) \]
\[ x(3) \rightarrow X(12) \]
\[ x(4) \rightarrow X(2) \]
\[ x(5) \rightarrow X(10) \]
\[ x(6) \rightarrow X(14) \]
\[ x(7) \rightarrow X(6) \]
\[ x(8) \rightarrow X(1) \]
\[ x(9) \rightarrow X(9) \]
\[ x(10) \rightarrow X(5) \]
\[ x(11) \rightarrow X(13) \]
\[ x(12) \rightarrow X(3) \]
\[ x(13) \rightarrow X(11) \]
\[ x(14) \rightarrow X(7) \]
\[ x(15) \rightarrow X(15) \]
16-point DIT
A Reordering Circuit is needed to perform the above conversion.