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
    \[
    A_k = \sum_{n=0}^{N/2-1} x_n W_N^{n2k} = \sum_{n=0}^{N/2-1} (x_n + x_{n+N/2}) W_N^{nk}
    \]
    \[
    A_{2k+1} = \sum_{n=0}^{N/4} x_n W_N^{n(2k+1)} = \sum_{n=0}^{N/4} (x_n - x_{n+N/2}) W_N^{nk} W_N^{n/2}
    \]

- 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

**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**

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

- **Radix-2 Algorithm**

- **DIF Radix-2 Algorithm**

\[
W_N^0 = -W_N^{N/2} = 1
\]

\[
W_N^{N/4} = -W_N^{N/4} = -j
\]

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

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

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

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

**Butterfly of Radix-2 Algorithm**

**DIF Form**

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

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

\[ k_j = 0, 1, \ldots, N/2 - 1. \]
FFT Algorithms

- Radix-4 Algorithm
  \[
  X(4k_l + l_i) = \sum_{k=0}^{N/4-1} [x(k) + x(n + N/4)N_{14} + x(n + N/2)N_{14} + x(n + 3N/4)N_{14}] W_N^{jl} + W_N^{nk}
  \]
  \[
  l = 0, 1, 2, 3; k_i = 0 \sim N/4 - 1;
  \]

- Radix-2^2 Algorithm
  \[
  X(4k_l + 2l_i + l_j)
  \]
  \[
  = \sum_{k=0}^{N/4-1} [x(n) + x(n + N/4)N_{14} + x(n + N/2)N_{14} + x(n + 3N/4)N_{14}] W_N^{jl} + W_N^{nk}
  \]
  \[
  = \sum_{k=0}^{N/4-1} [x(n) + x(n + N/2)] + x(n + N/4)] W_N^{jl} + W_N^{nk}
  \]
  \[
  l_i, l_j = 0, 1; k_i = 0 \sim N/4 - 1.
  \]

- Butterfly of Radix-4 Algorithm

(Data Ordering: Digit Reversed)
FFT Algorithms

Data Ordering of Radix-4 (N=16)

Butterfly of radix-2^2 Algorithm

(Data Ordering: Bit Reversed)
**FFT Algorithms**

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

\[
x(k_3,k_2,k_1,k_0) \quad X(k_0,k_1,k_2,k_3)
\]

- Bit-reversed ordering

**FFT Algorithms**

- DIF Radix-8 Algorithm

\[
X(8k + l) = \sum_{n=0}^{N-1} x(n)W_N^{-(l \times n)} = \sum_{n=0}^{N-1} x(n) + \frac{mN}{8}W_N^{-(l \times n)}
\]

\[
= \sum_{n=0}^{N-1} \sum_{m=0}^{3} [x(n) + (n + \frac{mN}{8})W_N^{-(l \times n)}]W_N^{-(l \times n)}
\]

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

\[
+ [x(n + \frac{N}{8}) + x(n + \frac{3N}{8})W_N^{-(l \times n)} + x(n + \frac{5N}{8})W_N^{-(l \times n)} + x(n + \frac{7N}{8})W_N^{-(l \times n)}]W_N^{-(l \times n)}
\]

\[l = 0,1,2,3,4,5,6,7; k = 0 - N / 8 - 1.\]
### FFT Algorithms

**DIF Radix-2³ Algorithm**

\[
X(8k + 4l_3 + 2l_2 + l_1)
\]

\[
= \sum_{n=0}^{N/8} \left\{ [x(n) + x(n + \frac{2N}{8})]W^n_8 + x(n + \frac{4N}{8})W^{2n}_8 + x(n + \frac{6N}{8})W^{3n}_8 \right\}
\]

\[
+ [x(n + \frac{N}{8}) + x(n + \frac{3N}{8})]W^n_8 + x(n + \frac{5N}{8})W^{2n}_8 + x(n + \frac{7N}{8})W^{3n}_8W_N^nW_N^{3n}W_N^{5n}
\]

\[
= \sum_{n=0}^{N/8} \left\{ [x(n) + W_8^n x(n + \frac{4N}{8}) + W_8^{2n} x(n + \frac{2N}{8}) + W_8^{3n} x(n + \frac{6N}{8})] \right\}
\]

\[
+ [x(n + \frac{N}{8}) + W_8^n x(n + \frac{5N}{8}) + W_8^{2n} x(n + \frac{3N}{8}) + W_8^{3n} x(n + \frac{7N}{8})]W_8^{2n}W_8^{3n}W_8^{5n}W_N^nW_N^{3n}W_N^{5n}W_N^{7n}
\]

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

**Butterfly of Radix-8 Algorithm**
FFT Algorithms

Butterfly of Radix-2$^3$ Algorithm

![Butterfly Diagram](image)

DIF Split-Radix 2/4 Algorithm

\[
X(2k) = \sum_{n=0}^{N/4-1} (x(n) + x(n + 2N/4))W_N^{nk}
\]

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

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

$k$ 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*

\[
X(2k) = \sum_{n=0}^{N/2-1} [x(n) + x(n + \frac{2N}{4})]W_N^{2nk}
\]

\[
X(8k + l) = \sum_{n=0}^{N/4-1} \left\{ [x(n) + x(n + \frac{2N}{8})W_4^l] + x(n + \frac{4N}{8})W_4^{2l} + x(n + \frac{6N}{8})W_4^{3l} \right\}
\]

\[
X(8k + l) = [x(n + \frac{N}{8}) + x(n + \frac{3N}{8})W_4^l] + x(n + \frac{5N}{8})W_4^{2l} + x(n + \frac{7N}{8})W_4^{3l}W_N^{2nk/N}
\]

\[
l = 1, 3, 5, 7
\]

**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: \( 1, j, (1+j)/\sqrt{2} \) (partially) removed
    - Radix-2/8: \( 1, j, (1+j)/\sqrt{2} \) removed

Radix-4 Signal Flow Graph
**Split-Radix Signal Flow Graph**

**Multiplicative Complexity**

<table>
<thead>
<tr>
<th>N</th>
<th>Radix-2</th>
<th>Radix-4</th>
<th>Split-Radix</th>
<th>Radix-8</th>
<th>Const. Mul</th>
<th>Radix-2/8</th>
<th>Const. Mul</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>8</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>32</td>
<td>34</td>
<td>31</td>
<td>26</td>
<td>20</td>
<td>8</td>
<td>16</td>
<td>14</td>
</tr>
<tr>
<td>64</td>
<td>98</td>
<td>76</td>
<td>72</td>
<td>48</td>
<td>32</td>
<td>44</td>
<td>38</td>
</tr>
<tr>
<td>128</td>
<td>258</td>
<td>215</td>
<td>186</td>
<td>152</td>
<td>64</td>
<td>120</td>
<td>94</td>
</tr>
<tr>
<td>256</td>
<td>642</td>
<td>492</td>
<td>456</td>
<td>376</td>
<td>128</td>
<td>308</td>
<td>214</td>
</tr>
<tr>
<td>512</td>
<td>1538</td>
<td>1239</td>
<td>1082</td>
<td>824</td>
<td>384</td>
<td>736</td>
<td>494</td>
</tr>
<tr>
<td>1024</td>
<td>3586</td>
<td>2732</td>
<td>2504</td>
<td>2104</td>
<td>768</td>
<td>1724</td>
<td>1128</td>
</tr>
<tr>
<td>2048</td>
<td>8194</td>
<td>6487</td>
<td>5690</td>
<td>4792</td>
<td>1536</td>
<td>3976</td>
<td>2494</td>
</tr>
<tr>
<td>4096</td>
<td>18434</td>
<td>13996</td>
<td>12744</td>
<td>10168</td>
<td>4096</td>
<td>8964</td>
<td>5494</td>
</tr>
<tr>
<td>8192</td>
<td>40962</td>
<td>32087</td>
<td>28218</td>
<td>23992</td>
<td>8192</td>
<td>19952</td>
<td>12046</td>
</tr>
</tbody>
</table>

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)
**RrMDC**

**Input stage**

**k th stage**

\[
T_{\text{UBHC}} N_r - T_{\text{UBHC}} N_r^2 - N_r \leq \cdot \cdot \cdot
\]

\[
0.65 \cdot 0.3 \cdot 2 - k_r N_r^1 \leq \cdot \cdot \cdot
\]

**Delay Feedback**

- R2SDF
- R4SDF
- R2^2SDF
$R2SDF (N=16)$  \textit{Radix-2 Single-Path Delay Feedback}

$R2SDF (N=16)$ vs. $R4SDF (N=128)$
**Buffer Styles of pipeline architecture**

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

![Image of R2 delay-commutator](image)

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

![Image of R2 delay-feedback](image)

---

**Single PE Architecture**

![Image of Single PE Architecture](image)
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,N stages to compute N point FFT.
- Arbitrary radix can be used
- Base on Cooley-Tukey decomposition for any composite number

![Diagram of FFT Design](image-url)
**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

![Diagram](image)
**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.

\[
\begin{align*}
A_{2k} &= \sum_{n=0}^{N/2-1} (X_n + X_{n+N/2}) \cdot W_{N/2}^n \\
A_{4k+1} &= \sum_{n=0}^{N/4-1} (X_n - j \cdot X_{n+N/4} - X_{n+N/2} + j \cdot X_{n+3N/4}) \cdot W_{N/4}^n \cdot W_{N/2}^n \\
A_{4k+2} &= \sum_{n=0}^{N/4-1} (X_n + j \cdot X_{n+N/4} - X_{n+N/2} - j \cdot X_{n+3N/4}) \cdot W_{N/4}^n \cdot W_{N/2}^n
\end{align*}
\]
The Kernel of Processing Element

Folded Butterfly Units

- Comparing with Radix-2/Radix-2², 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
N=64, E=2, Radix-2 Cached-FFT