The Fast Fourier Transform

Sebastian Schüppel

Portland State University
Department of Electrical and Computer Engineering

02|04|2008
Use of FFT algorithm

- originally discovered by Gauß in 1804
- ’rediscovery’ by Danielson and Lanczos in 1942
- generally know through Cooley / Tukey in mid 60s
- DFT: $O(n^2)$ vs. FFT: $O(n \log_2(n))$

\[ X = \sum_{k=0}^{N-1} x_k e^{-i\left(\frac{2\pi}{N}\right)kn} \quad (1) \]

- use $N = 2^p$ number of samples
- divide and conquer algorithm
Twiddle Factor

\[ e^{-i \frac{2\pi}{N}} = W_N \]  

\[ X(n) = \sum_{k=0}^{N-1} x_k e^{-i \left( \frac{2\pi}{N} \right) kn} = \sum_{k=0}^{N-1} x_k W_N^{kn} \]

- Euler’s Law: \[ e^{i\theta} = \sin(\theta) + i \cos(\theta) = \sin\left(\frac{2\pi}{N}\right) - i \cos\left(\frac{2\pi}{N}\right) \]
Fast Fourier Transform

Divide and Conquer

\[ X(n) = \sum_{k=0}^{(N/2)-1} x_{2k} e^{-i\left(\frac{2\pi}{N}\right)(2k)n} + \sum_{k=0}^{(N/2)-1} x_{2k+1} e^{-i\left(\frac{2\pi}{N}\right)(2k+1)n} \]

\[ = \sum_{k=0}^{(N/2)-1} x_{2k} e^{-i\left(\frac{2\pi}{N}\right)(2k)n} + e^{-i\left(\frac{2\pi}{N}\right)n} \sum_{k=0}^{(N/2)-1} x_{2k+1} e^{-i\left(\frac{2\pi}{N}\right)(2k)n} \]

\[ = \sum_{k=0}^{M} x_{2k} W_{M}^{kn} + W_{N}^{n} \sum_{k=0}^{M} x_{2k+1} W_{M}^{kn} \quad \left( M = \frac{N}{2} \right) \quad (4) \]

Solution through recursion

\[ N = 1 \quad \Rightarrow \quad X(0) = x_0 W_{1}^{0} = x_0 \]
Fast Fourier Transform

Basic Building Block - Butterfly

\[ X(n) = X_{2k} + W_n^k X_{2k+1} \quad \left( M = \frac{N}{2} \right) \]  

\[ X(n + M) = X_{2k} - W_n^k X_{2k+1} \]
Fast Fourier Transform - $N = 16$

<table>
<thead>
<tr>
<th>0</th>
<th>+</th>
<th>+</th>
<th>+</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>15</td>
</tr>
</tbody>
</table>

bit-reversed order!
FFT IP-Cores

Why using IP-Cores?
- Easy to Implement
- Free (Altera)
- Optimized in Speed and Area

Altera Features
- Matlab Model
- radix-4 and mixed radix-4/2
- different Input and Output orders
- transformation length $2^m \ 6 \leq m \leq 14$
Inside of the Altera Core

Single and Quad radix-4 Output
"Fourier Transform for Pedestrians" published by Springer Verlag