11 DSP Algorithm Implementation

Introduction

Two types of algorithms:

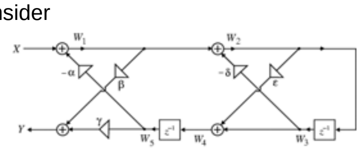
1) Filtering algorithms
2) Signal analysis algorithms

Basic implementation approaches:

1) Hardware
2) Firmware
3) Software

Matrix Representation of Digital Filter Structures

- A digital filter structure can be described in the time-domain by a set of equations relating the output sequence to the input sequence and, in some cases, one or more internally generated sequences
- Consider



Matrix Representation of Digital Filter Structures

- This structure, in the time-domain, is described by the set of equations

$$w_1[n] = x[n] - \alpha\, w_5[n]$$
$$w_2[n] = w_1[n] - \delta\, w_3[n]$$
$$w_3[n] = w_2[n-1]$$
$$w_4[n] = w_3[n] + \varepsilon\, w_2[n]$$
$$w_5[n] = w_4[n-1]$$
$$y[n] = \beta\, w_1[n] + \gamma\, w_5[n]$$

Matrix Representation of Digital Filter Structures

- The equations cannot be implemented in the order shown with each variable on the left side computed before the variable below is computed
- For example, computation of $w_1[n]$ in the 1st step requires the knowledge of $w_5[n]$ which is computed in the 5th step
- Likewise, computation of $w_2[n]$ in the 2nd step requires the knowledge of $w_3[n]$ that is computed in the 3rd step

Matrix Representation of Digital Filter Structures

- This ordered set of equations is said to be **noncomputable**
- Suppose we reorder these equations

$$w_3[n] = w_2[n-1]$$
$$w_5[n] = w_4[n-1]$$
$$w_1[n] = x[n] - \alpha\, w_5[n]$$
$$w_2[n] = w_1[n] - \delta\, w_3[n]$$
$$y[n] = \beta\, w_1[n] + \gamma\, w_5[n]$$
$$w_4[n] = w_3[n] + \varepsilon\, w_2[n]$$

Matrix Representation of Digital Filter Structures

- This ordered set of equations is **computable**
- In most practical applications, equations describing a digital filter structure can be put into a computable order by inspection
- A simple way to examine the computability of equations describing a digital filter structure is by writing the equations in a matrix form

Matrix Representation

- A matrix representation of the first ordered set of equations is

$$\begin{bmatrix} w_1[n] \\ w_2[n] \\ w_3[n] \\ w_4[n] \\ w_5[n] \\ y[n] \end{bmatrix} = \begin{bmatrix} x[n] \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & -\alpha & 0 \\ 1 & 0 & -\delta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \varepsilon & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \beta & 0 & 0 & 0 & \gamma & 0 \end{bmatrix} \begin{bmatrix} w_1[n] \\ w_2[n] \\ w_3[n] \\ w_4[n] \\ w_5[n] \\ y[n] \end{bmatrix}$$

$$+ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} w_1[n-1] \\ w_2[n-1] \\ w_3[n-1] \\ w_4[n-1] \\ w_5[n-1] \\ y[n-1] \end{bmatrix}$$

Matrix Representation

- In compact form

$$\mathbf{y}[n] = \mathbf{x}[n] + \mathbf{F}\,\mathbf{y}[n] + \mathbf{G}\,\mathbf{y}[n-1]$$

where

$$\mathbf{y}[n] = [w_1[n] \quad w_2[n] \quad w_3[n] \quad w_4[n] \quad w_5[n] \quad y[n]]^T$$

$$\mathbf{x}[n] = [x[n] \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]^T$$

$$\mathbf{F} = \begin{bmatrix} 0 & 0 & 0 & 0 & -\alpha & 0 \\ 1 & 0 & -\delta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \varepsilon & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \beta & 0 & 0 & 0 & \gamma & 0 \end{bmatrix}, \quad \mathbf{G} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Matrix Representation

- For the computation of present value of a particular signal variable, nonzero entries in the corresponding rows of matrices $\mathbf{F}$ and $\mathbf{G}$ determine the variables whose present and previous values are needed
- If a diagonal element of $\mathbf{F}$ is nonzero, then computation of present value of the corresponding variable requires the knowledge of its present value ***implying presence of a delay-free loop***

Matrix Representation

- Any nonzero entries in the same row above the main diagonal of $\mathbf{F}$ imply that the computation of present value of the corresponding variable requires present values of other variables not yet computed, making the set of equations noncomputable
- Hence, ***for computability all elements of $\mathbf{F}$ matrix on the diagonal and above diagonal must be zeros***

Matrix Representation

- In the $\mathbf{F}$ matrix for the first ordered set of equations, diagonal elements are all zeros, indicating absence of delay-free loops
- However, there are nonzero entries above the diagonal in the first and second rows of $\mathbf{F}$ indicating that the set of equations are not in proper order for computation

Matrix Representation

- The **F** matrix for the second ordered set of equations is

$$\mathbf{F} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\alpha & 0 & 0 & 0 & 0 \\ -\delta & 0 & 1 & 0 & 0 & 0 \\ 0 & \gamma & \beta & 0 & 0 & 0 \\ 1 & 0 & 0 & \varepsilon & 0 & 0 \end{bmatrix}$$

which is seen to satisfy the computability condition

Precedence Graph

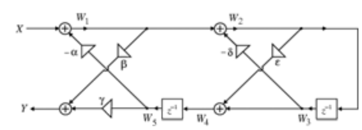
- The **precedence graph** can be used to test the computability of a digital filter structure and to develop the proper ordering sequence for a set of equations describing a computable structure
- It is developed from the **signal-flow graph** description of the digital filter structure in which independent and dependent signal variables are represented by **nodes**, and the multiplier and delay branches are represented by **directed branches**

Precedence Graph

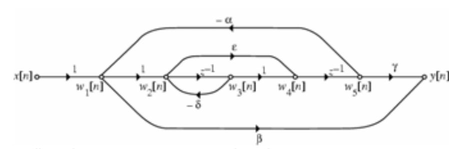
- The directed branch has an attached symbol denoting the **branch gain** or **transmittance**
- For a multiplier branch, the branch gain is the multiplier coefficient value
- For a delay branch, the branch gain is simply z^{-1}

Precedence Graph

- The signal-flow graph representation of

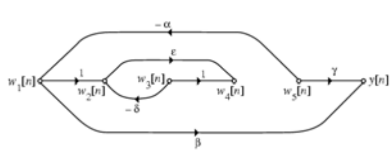


is shown below



Precedence Graph

- A **reduced signal-flow graph** is then developed by removing the delay branches and all branches going out of the input node
- The reduced signal-flow graph of the example digital filter structure is shown below



Precedence Graph

- The remaining nodes in the reduced signal-flow graph are grouped as follows:
- All nodes with only outgoing branches are grouped into one set labeled $\{N_1\}$
- Next, the set $\{N_2\}$ is formed containing nodes coming in only from one or more nodes in the set $\{N_1\}$ and have outgoing branches to the other nodes

Precedence Graph

- Then, form the set $\{N_3\}$ containing nodes that have branches coming in only from one or more nodes in the sets $\{N_1\}$ and $\{N_2\}$, and have outgoing branches to other nodes
- Continue the process until there is a set of nodes $\{N_f\}$ containing only incoming branches
- The rearranged signal-flow graph is called a **precedence graph**

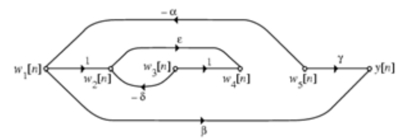
Precedence Graph

- Since signal variables belonging to $\{N_1\}$ do not depend on the present values of other signal variables, these variables should be computed first
- Next, signal variables belonging to $\{N_2\}$ can be computed since they depend on the present values of signal variables contained in $\{N_1\}$that have already been computed

Precedence Graph

- This is followed by the computation of signal variables in $\{N_3\}$, $\{N_4\}$, etc.
- Finally, in the last step the signal variables in $\{N_f\}$ are computed
- This process of sequential computation ensures the development of a valid computational algorithm
- If there is no final set $\{N_f\}$ containing only incoming branches, the digital filter structure is noncomputable

Precedence Graph

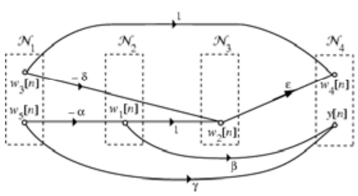


- For the example precedence graph, pertinent groupings of node variables are:

$$\{N_1\} = \{w_3[n], w_5[n]\}$$
$$\{N_2\} = \{w_1[n]\}$$
$$\{N_3\} = \{w_2[n]\}$$
$$\{N_4\} = \{w_4[n], y[n]\}$$

Precedence Graph

- Precedence graph redrawn according to the groupings based on ordering computations is shown below



- Since the final node set $\{N_4\}$ has only incoming branches, the structure is computable

Efficient Algorithms for the DFT Computation

- The discrete Fourier transform (DFT) is a widely used DSP algorithm
  - It can be used to implement the linear convolution of two sequences, a key digital filtering operation
  - It is also used in spectral analysis of signals
- Because of the widespread use of DFT, it is of interest to investigate efficient implementation methods of the DFT
- Various approaches are available:
  - Fast Fourier Transform (FFT) algorithm was invented in 1965 by J.W. Cooley and J.W. Tukey

Computation of the DFT

- N-point DFT $X[k]$ of a sequence $x[n]$ of length N

$$X[k] = X(e^{j\omega})\Big|_{\omega=2\pi k/N} = \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N}, \quad k = 0,1,\ldots,N-1$$

- DFT gives N samples of the Fourier transform evaluated uniformly on the ω-axis at $\omega_k=2\pi k/N$, $0 \le k \le N-1$
- DFT gives the samples of $X(z)$ of the sequence $x[n]$ on the unit circle

$$X[k] = X(z)\Big|_{z=e^{j2\pi k/N}}, \quad k = 0,1,\ldots,N-1$$

Computation of the DFT

Computational complexity of $X[k]$:

- Computation of each sample requires N complex multiplications and N-1 complex additions
- Computation of N samples require N^2 complex multiplications and $N(N-1)$ complex additions

  If N is large, the computation of the DFT requires N^2 complex operations

  (where a complex operation corresponds to one complex multiplication and one complex addition)

Decimation in Time FFT

- The sequence $x[n]$ of length N is separated into two sequences of length $N/2$ composed of the even and odd indexed samples:

$$\begin{aligned} X[k] &= \sum_{n=0}^{N-1} x[n]W_N^{kn} = \sum_{n=even} x[n]W_N^{kn} + \sum_{n=odd} x[n]W_N^{kn} \\ &= \sum_{m=0}^{(N/2)-1} x[2m]W_N^{k2m} + \sum_{m=0}^{(N/2)-1} x[2m+1]W_N^{k(2m+1)} \\ &= \sum_{m=0}^{(N/2)-1} x[2m]\left(W_N^2\right)^{km} + W_N^k \sum_{m=0}^{(N/2)-1} x[2m+1]\left(W_N^2\right)^{km} \end{aligned}$$

- Notice that: $W_N^2 = e^{-2j(2\pi/N)} = e^{-j2\pi/(N/2)}$

Decimation in Time FFT

- The two DFTs are true $(N/2)$-point DFTs of the even- and odd-indexed parts of the original $x[n]$:

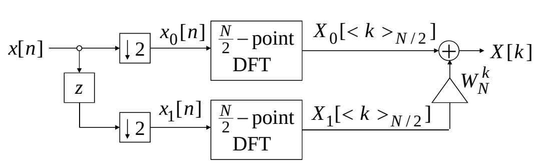
$$X[k] = \sum_{m=0}^{(N/2)-1} x[2m]\left(W_N^2\right)^{km} + W_N^k \sum_{m=0}^{(N/2)-1} x[2m+1]\left(W_N^2\right)^{km}$$

$$\Rightarrow \boxed{X[k] = X_e[k] + W_N^k X_o[k]}$$

- This basic idea can be applied again and again until only two-point DFTs are left, i.e., each $(N/2)$-point DFT is computed by combining two $(N/4)$-point DFTs, each of which is computed by combining two $(N/8)$-point DFTs, etc.

Decimation-in-Time FFT Algorithm

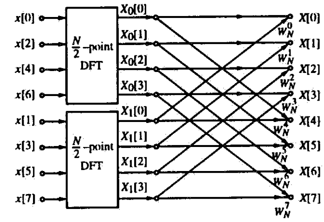
- Block-diagram interpretation



Notice! $X_0[\langle k \rangle_{N/2}] \Leftrightarrow X_e(k)$ and $X_1[\langle k \rangle_{N/2}] \Leftrightarrow X_o(k)$

Decimation in Time FFT

$$\boxed{X[k] = X_e[k] + W_N^k X_o[k]}$$



Decimation in Time FFT

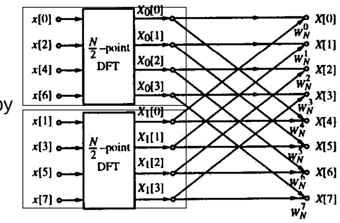
$$\boxed{X[k] \quad = X_e[k] + W_N^k X_o[k]}$$

- Direct DFT computation: N^2 complex operations where complex operation consists of one complex multiplication and one complex addition
- FFT after one decomposition: $2(N/2)^2 + N$ complex operations, i.e. two $N/2$ –point DFTs and combining their results
- Example $N = 8 = 2^3$:
  - $N^2 = 64$ and
  - $2(N/2)^2 + N = 2*16 + 8 = 40$

Decimation in Time FFT

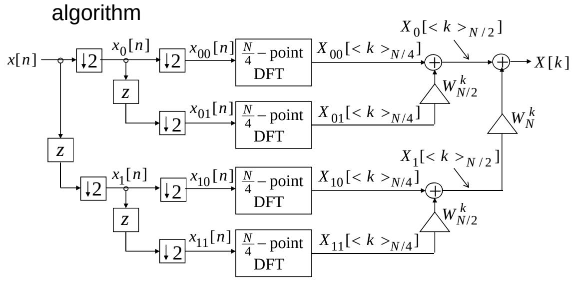
- Next, the $N/2$-point DFTs are decomposed into two $N/4$-point DFTs resulting in altogether four $N/4$-point DFTs

$N/2$-point DFTs are replaced by $N/4$-point DFTs



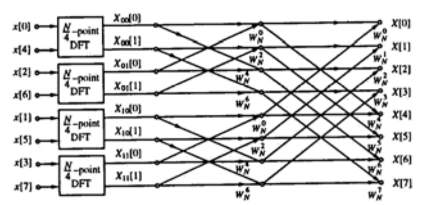
Decimation-in-Time FFT Algorithm

- Block-diagram representation of the two-stage algorithm



Decimation in Time FFT

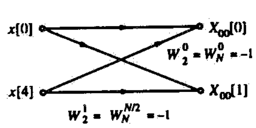
- Flow graph of the second stage in decimation-in-time FFT algorithm for N=8



- # operations: $2\left(\frac{N}{2}\right)^2 + N \;\Rightarrow\; 2\left[2\left(\frac{N}{4}\right)^2 + \frac{N}{2}\right] + N = 4\left(\frac{N}{4}\right)^2 + N + N$

Decimation in Time FFT

- Finally, the flow graph of the basic 2-point DFT is:



$= x[0] + x[1]$

$= x[0] - x[1]$

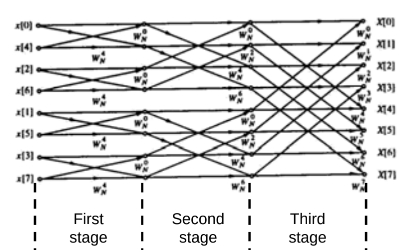
$$W_N^{N/2} = e^{-j(2\pi/N)(N/2)} = e^{-j\pi} = -1$$

$$W_2^k = W_N^{(N/2)k} = -1$$

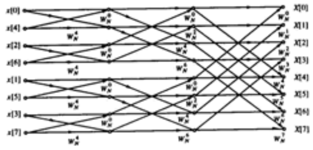
$$X_{00}[k] = \sum_{n=0}^{1} x_{00}[n] W_2^{nk} = x[0] + W_2^k x[4]$$

Decimation in Time FFT

- Complete flow graph of the basic decimation-in-time FFT algorithm for N=8



Decimation in Time FFT

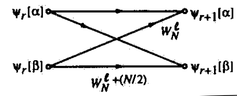


$$N = 2^{\mu},$$
$$\mu = \log_2 N$$

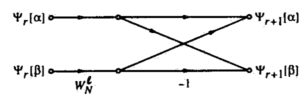
- The DFT consists of μ stages
- Each stage consists of $N/2$ basic computational modules, called "butterflies"
- Each butterfly contains two complex operations, i.e., two complex multiplications and two complex additions

$\Rightarrow$ $N \log_2 N$ ***operations***

Butterfly Computation



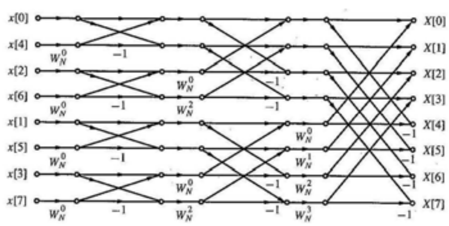
$$\Psi_{r+1}[\alpha] = \Psi_r[\alpha] + W_N^l \Psi_r[\beta]$$
$$\Psi_{r+1}[\beta] = \Psi_r[\alpha] + \underbrace{W_N^{l+(N/2)}}\Psi_r[\beta]$$
$$W_N^{l+(N/2)} = W_N^l W_N^{(N/2)} = -W_N^l$$
$$\Rightarrow \Psi_{r+1}[\beta] = \Psi_r[\alpha] - W_N^l \Psi_r[\beta]$$



- Each butterfly can be realized with only one complex multiplication (and two additions)

Decimation in Time FFT

- Flow graph of the modified DIT FFT algorithm



- Complexity: $(N/2)\log_2 N$ complex multiplications
  $N\log_2 N$ complex additions

Properties of the DIT FFT Algorithms

- ***In-place computation* =>** Efficient memory utilization
- ***Bit-reversed ordering* :**

✓Input sequence $x[m]$ in normal order;

✓Location index n is obtained by reversing the order of bits in the index

m	n	Input	New location
000	000	x[0]	Location 0
001	100	x[1]	Location 4
010	010	x[2]	Location 2
011	110	x[3]	Location 6
100	001	x[4]	Location 1
101	101	x[5]	Location 5
110	011	x[6]	Location 3
111	111	x[7]	Location 7

Decimation in Frequency FFT

- The length-N sequence $x[n]$ is decomposed sequentially into sets of smaller and smaller subsequences
- The DFT is formed as a weighted combination of the DFTs of these subsequences
- Dividing first $x[n]$ into two sequences of length $N/2$:

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

Decimation in Frequency FFT

- Using the identity $W_N^{(N/2)k} = (-1)^k$
- The DFT can be rewritten as

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

- For k even: $X[2l] = \sum_{n=0}^{(N/2)-1} [x[n] + x[n+(N/2)]] W_N^{2nl}$

$$= \sum_{n=0}^{(N/2)-1} [x[n] + x[n+(N/2)]] W_{N/2}^{nl} \quad 0 \le l \le \frac{N}{2} - 1$$

- For k odd: $X[2l+1] = \sum_{n=0}^{(N/2)-1} [x[n] - x[n+(N/2)]] W_N^{n(2l+1)}$

$$= \sum_{n=0}^{(N/2)-1} [x[n] - x[n+(N/2)]] W_N^{n} W_{N/2}^{nl} \quad 0 \le l \le \frac{N}{2} - 1$$

Decimation in Frequency FFT

- The $N/2$-point sequences $x_0[n]$ and $x_1[n]$ are obtained as the sum and difference of the first and second half of the original sequence $x[n]$:

$$x_0[n] = \left\{ x[n] + x\left[\frac{N}{2}+n\right] \right\}$$

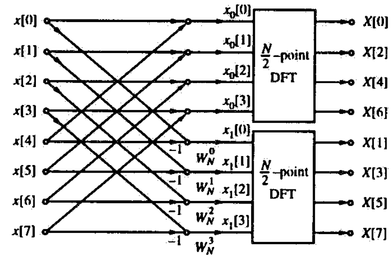
$$x_1[n] = \left\{ x[n] - x\left[\frac{N}{2}+n\right] \right\} W_N^n$$

$$for \quad n = 0,1,\ldots,\frac{N}{2}-1$$

- In the first stage of the decimation in frequency FFT, true $N/2$-point DFTs of the above sequences are computed

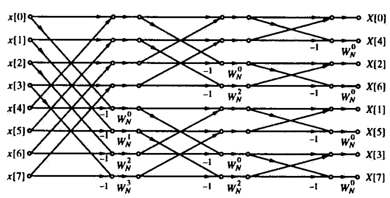
Decimation in Frequency FFT

- The flow graph of the first stage of the decimation-in-frequency FFT algorithm for N=8:



Decimation in Frequency FFT

- Decomposing the $N/2$-point, $N/4$-point, …, DFTs into $N/4$-point, $N/8$-point, …, DFTs results in the decimation-in-frequency FFT algorithm

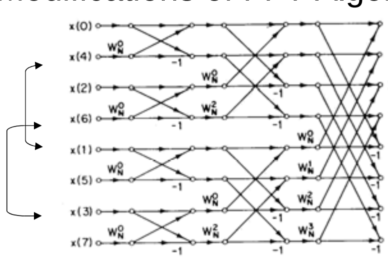


- The complexity as well as other properties (in-place computation, indexing) are similar to the DIT FFT

Comparison of DIT and DIF FFT Algorithms

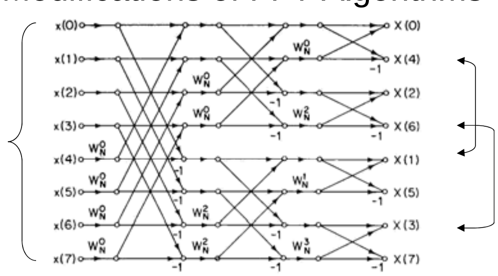
- DIT algorithm: Input in bit-reversed order and output in normal order
- DIF algorithm: Input in normal order and output in order bit-reversed
- DIT and DIF algorithms are both in-place
- DIT and DIF algorithms can be obtained from each other using flow reversal in the structure
- The computational complexity is the same in DIT and DIF algorithms

Modifications of FFT Algorithms



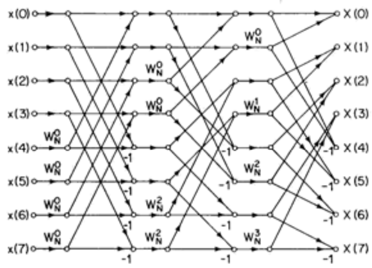
- The order of input and output data samples can be changed by interchanging rows $x[4]$ and $x[1]$ as well as rows $x[6]$ and $x[3]$

Modifications of FFT Algorithms



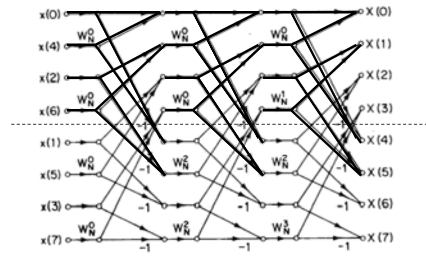
- The input will be in normal order
- The output will be in bit-reversed order

Modifications of FFT Algorithms



- Rearrangement of the DIT FFT algorithm with both input and output in normal order

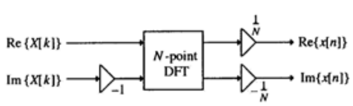
Modifications of FFT Algorithms



- Rearrangement of the DIT FFT algorithm having the same geometry for each stage, thereby permitting sequential data accessing and storage

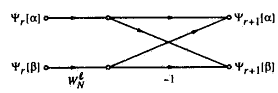
Inverse DFT Computation

- The sequence $x[n]$ is related to the samples of $X[k]$ through: $x[n]=\frac{1}{N}\sum_{k=0}^{N-1}X[k]W_N^{-nk}$
- Multiplying by N and taking the complex conjugate: $Nx^*[n]=\sum_{k=0}^{N-1}X^*[k]W_N^{nk}$
- The desired IDFT $x[n]$ is then obtained as: $x[n]=\frac{1}{N}\left\{\sum_{k=0}^{N-1}X^*[k]W_N^{nk}\right\}^*$
- Realization via FFT:



Inverse DFT Computation

- The basic DIT butterfly is:



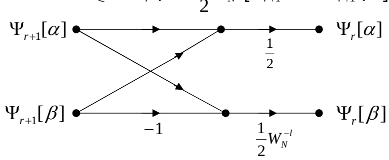
$$\begin{cases}\Psi_{r+1}[\alpha]=\Psi_r[\alpha]+W_N^{l}\Psi_r[\beta]\\ \Psi_{r+1}[\beta]=\Psi_r[\alpha]-W_N^{l}\Psi_r[\beta]\end{cases}$$

- The IFFT algorithm can be derived by inverting the basic butterfly computation
- Solving $\Psi_r[\alpha]$ and $\Psi_r[\beta]$ using $\Psi_{r+1}[\alpha]$ and $\Psi_{r+1}[\beta]$ yields:

$$\begin{cases}2\Psi_r[\alpha]=\Psi_{r+1}[\alpha]+\Psi_{r+1}[\beta]\\ 2W_N^{l}\Psi_r[\beta]=\Psi_{r+1}[\alpha]-\Psi_{r+1}[\beta]\end{cases}$$

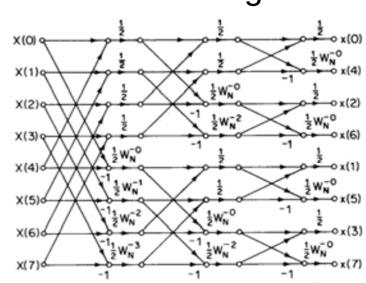
Inverse DFT Computation

$$\begin{cases}\Psi_r[\alpha]=\frac{1}{2}\left[\Psi_{r+1}[\alpha]+\Psi_{r+1}[\beta]\right]\\ \Psi_r[\beta]=\frac{1}{2}W_N^{-l}\left[\Psi_{r+1}[\alpha]-\Psi_{r+1}[\beta]\right]\end{cases}$$



- The DIF IFFT butterfly is obtained by flow-reversal from the DIT FFT butterfly

Inverse FFT Algorithm



- The DIF IFFT algorithm has been obtained from the DIT FFT algorithm by flow-reversal, i.e., as transpose

Fast DFT Algorithms Based on Index Mapping

- Usually, fast DFT algorithms are for sequences of length N that is a power-of-2 integer
- For the case when the length N of the sequence is a composite number that is expressible as a product of integers, it is possible to develop computationally fast DFT algorithms via index mapping where the sample indices n and k are mapped into two-dimensional indices
- The algorithms compute the length-N DFT through a series of smaller length DFTs

Fast DFT Algorithms Based on Index Mapping

- The Cooley-Tukey FFT algorithm can be generalized for composite N
- Even more efficient DFT algorithms can be obtained for N that is expressible as a product of prime numbers

$\Rightarrow$ ***Prime factor FFT algorithms***