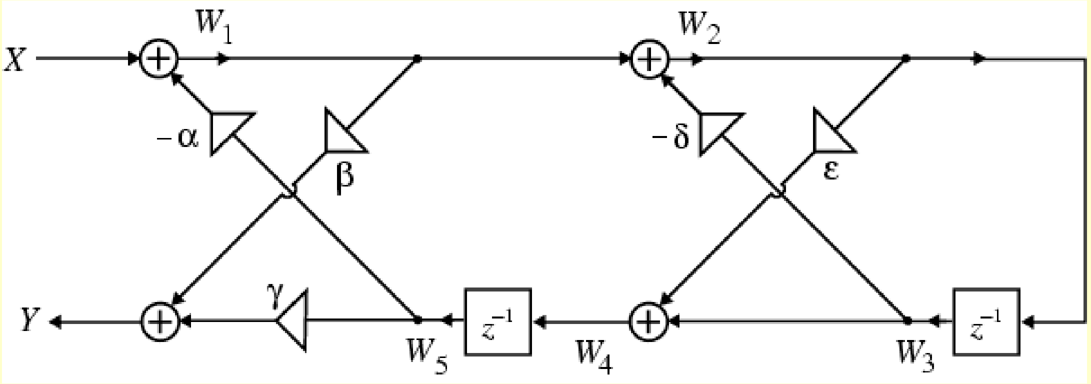
Matrix Representation of Digital Filter Structures

- 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

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

where

$$\mathbf{y}[n] = \begin{bmatrix} w_1[n] & w_2[n] & w_3[n] & w_4[n] & w_5[n] & y[n] \end{bmatrix}^T$$

$$\mathbf{x}[n] = \begin{bmatrix} x[n] & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

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

Matrix Representation

- For the computation of present value of a particular signal variable, nonzero entries in the corresponding rows of matrices **F** and **G** determine the variables whose present and previous values are needed
- If a diagonal element of **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 **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 **F** matrix on the diagonal and above diagonal must be zeros

Matrix Representation

- In the **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 **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

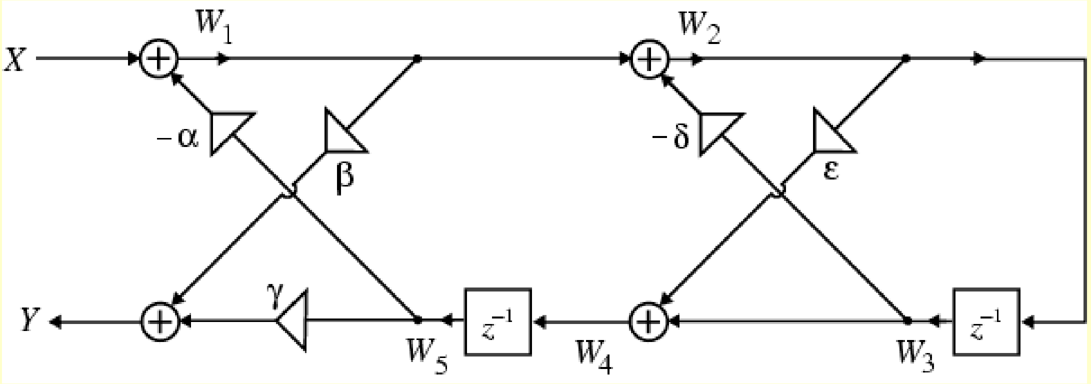
- The **precedence graph** can be used to test the computability of a digital filter structure and to develop the proper ordering sequence for a set of equations describing a computable structure
- It is developed from the **signal-flow graph** description of the digital filter structure in which independent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

Precedence Graph

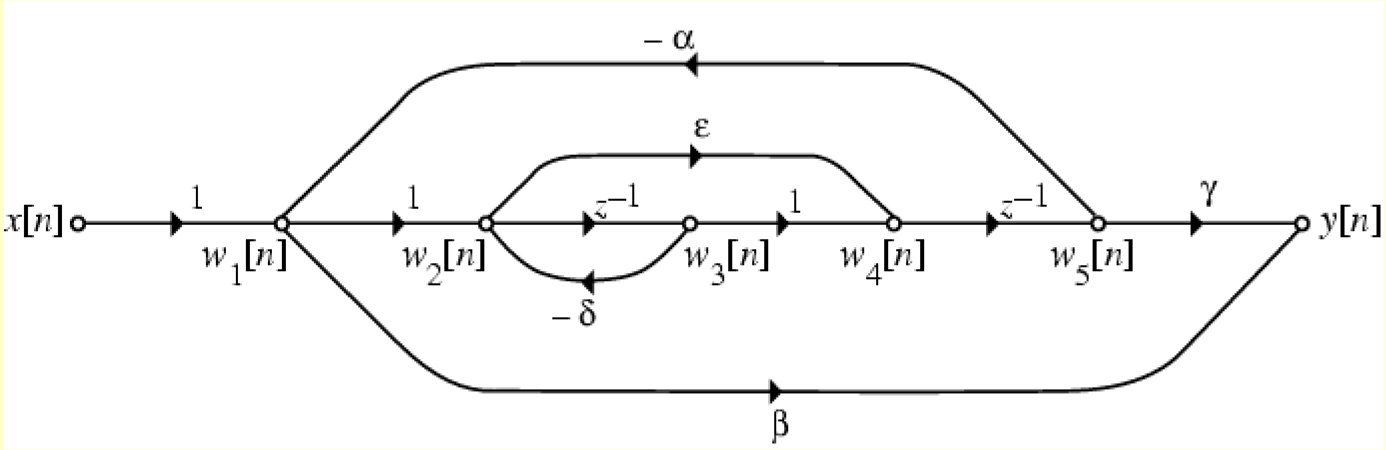
- The directed branch has an attached symbol denoting the **branch gain** or **transmittance**
- For a multiplier branch, the branch gain is the multiplier coefficient value
- For a delay branch, the branch gain is simply z^{-1}

Precedence Graph

- The signal-flow graph representation of

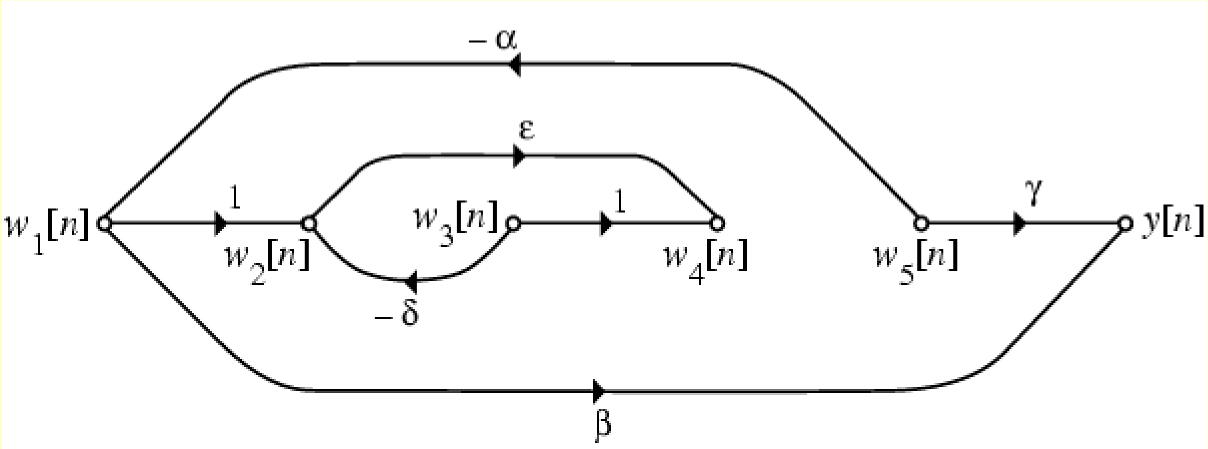


is shown below



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 $\{\mathcal{N}_1\}$
- Next, the set $\{\mathcal{N}_2\}$ is formed containing nodes coming in only from one or more nodes in the set $\{\mathcal{N}_1\}$ and have outgoing branches to the other nodes

Precedence Graph

- Then, form the set $\{\mathcal{N}_3\}$ containing nodes that have branches coming in only from one or more nodes in the sets $\{\mathcal{N}_1\}$ and $\{\mathcal{N}_2\}$, and have outgoing branches to other nodes
- Continue the process until there is a set of nodes $\{\mathcal{N}_f\}$ containing only incoming branches
- The rearranged signal-flow graph is called a **precedence graph**

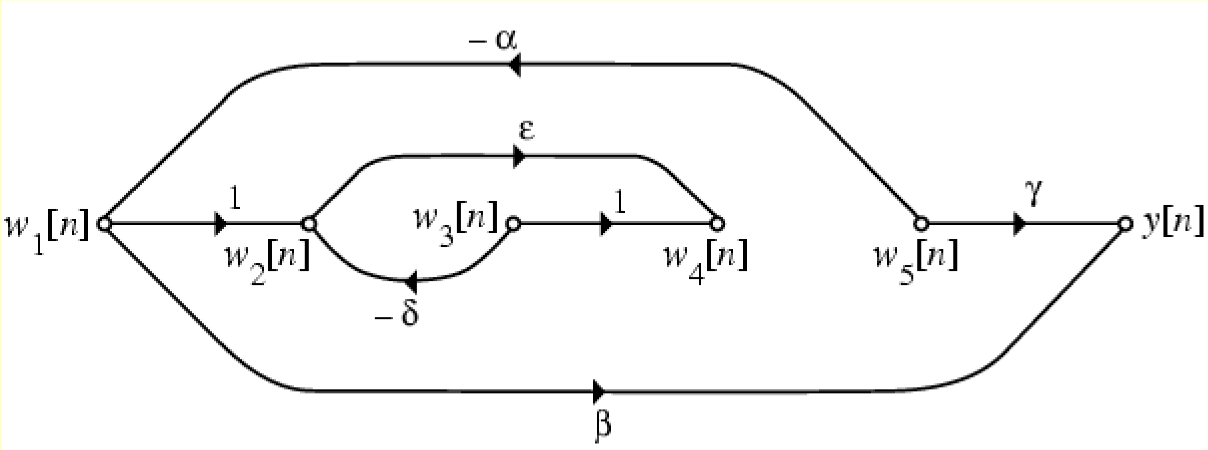
Precedence Graph

- Since signal variables belonging to $\{\mathcal{N}_1\}$ do not depend on the present values of other signal variables, these variables should be computed first
- Next, signal variables belonging to $\{\mathcal{N}_2\}$ can be computed since they depend on the present values of signal variables contained in $\{\mathcal{N}_1\}$ that have already been computed

Precedence Graph

- This is followed by the computation of signal variables in $\{\mathcal{N}_3\}$, $\{\mathcal{N}_4\}$, etc.
- Finally, in the last step the signal variables in $\{\mathcal{N}_f\}$ are computed
- This process of sequential computation ensures the development of a valid computational algorithm
- If there is no final set $\{\mathcal{N}_f\}$ containing only incoming branches, the digital filter structure is noncomputable

Precedence Graph



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

$$\{\mathcal{N}_1\} = \{w_3[n], w_5[n]\}$$

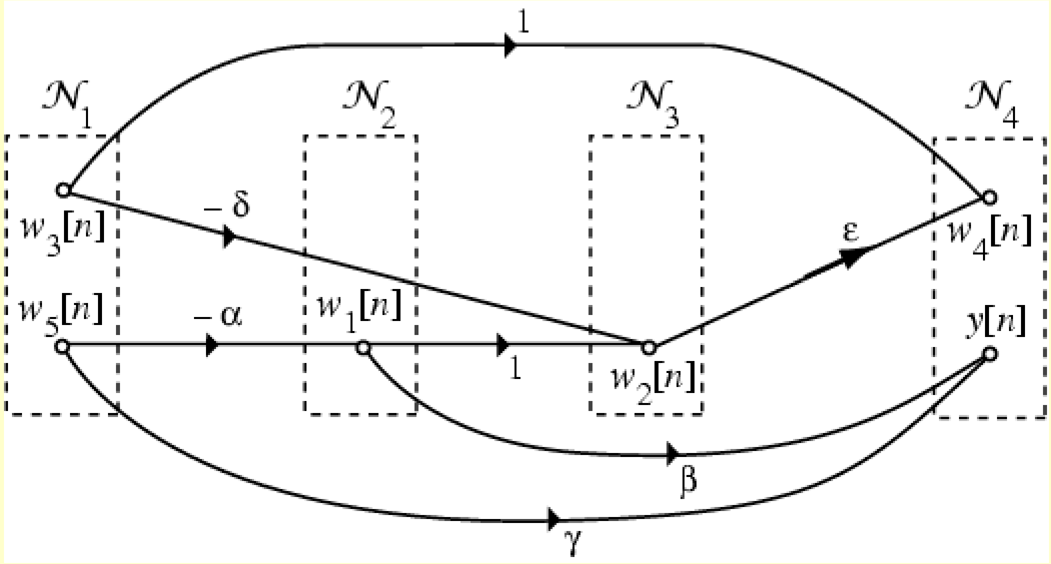
$$\{\mathcal{N}_2\} = \{w_1[n]\}$$

$$\{\mathcal{N}_3\} = \{w_2[n]\}$$

$$\{\mathcal{N}_4\} = \{w_4[n], y[n]\}$$

Precedence Graph

- Precedence graph redrawn according to the above groupings is as shown below



- Since the final node set $\{\mathcal{N}_4\}$ has only incoming branches, the structure is computable

Structure Verification

- A simple method to verify that the structure developed is indeed characterized by the prescribed transfer function $H(z)$
- Consider for simplicity a causal 3rd order IIR transfer function

$$H(z) = \frac{P(z)}{D(z)} = \frac{p_0 + p_1 z^{-1} + p_2 z^{-2} + p_3 z^{-3}}{1 + d_1 z^{-1} + d_2 z^{-2} + d_3 z^{-3}}$$

- If $\{h[n]\}$ denotes its impulse response, then

$$H(z) = \sum_{n=0}^{\infty} h[n] z^{-n}$$

Structure Verification

- Note $P(z) = H(z)D(z)$

  which is equivalent to $p_n = \sum_{k=0}^{n} h[k]d_{n-k}, d_0 = 1$

- Evaluate above convolution sum for $0 \le n \le 6$:

$$p_0 = h[0]$$
$$p_1 = h[1] + h[0]d_1$$
$$p_2 = h[2] + h[1]d_1 + h[0]d_2$$
$$p_3 = h[3] + h[2]d_1 + h[1]d_2 + h[0]d_3$$
$$0 = h[4] + h[3]d_1 + h[2]d_2 + h[1]d_3$$
$$0 = h[5] + h[4]d_1 + h[3]d_2 + h[2]d_3$$
$$0 = h[6] + h[5]d_1 + h[4]d_2 + h[3]d_3$$

Structure Verification

- In matrix form we get

$$\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \\ \hdashline 0 \\ 0 \\ 0 \end{bmatrix} = \left[\begin{array}{c:ccc} h[0] & 0 & 0 & 0 \\ h[1] & h[0] & 0 & 0 \\ h[2] & h[1] & h[0] & 0 \\ h[3] & h[2] & h[1] & h[0] \\ \hdashline h[4] & h[3] & h[2] & h[1] \\ h[5] & h[4] & h[3] & h[2] \\ h[6] & h[5] & h[4] & h[3] \end{array}\right] \begin{bmatrix} 1 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix}$$

- In partitioned form above matrix equation can be written as

$$\begin{bmatrix} \mathbf{p} \\ \cdots \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} & \mathbf{H}_1 & \\ \cdots & \cdots & \cdots \\ \mathbf{h} & \vdots & \mathbf{H}_2 \end{bmatrix} \begin{bmatrix} 1 \\ \cdots \\ \mathbf{d} \end{bmatrix}$$

Structure Verification

where

$$\mathbf{p} = \mathbf{H}_1 \begin{bmatrix} 1 \\ \mathbf{d} \end{bmatrix}, \qquad \mathbf{0} = [\mathbf{h} \quad \mathbf{H}_2] \begin{bmatrix} 1 \\ \mathbf{d} \end{bmatrix}$$

- Solving second equation we get

$$\mathbf{d} = -\mathbf{H}_2^{-1}\mathbf{h}$$

- Substituting above in the first equation we get

$$\mathbf{p} = \mathbf{H}_1 \begin{bmatrix} 1 \\ -\mathbf{H}_2^{-1}\mathbf{h} \end{bmatrix}$$

- In the case of an N-th order IIR filter, the coefficients of its transfer function can be determined from the first $2N+1$ impulse response samples

Structure Verification

- Example - Consider the causal transfer function

$$H(z)=\frac{2+6z^{-1}+3z^{-2}}{1+z^{-1}+2z^{-2}}=2+4z^{-1}-5z^{-2}-3z^{-3}+13z^{-4}+\cdots$$

- Here

$$h[0]=2,\ h[1]=4,\ h[2]=-5,\ h[3]=-3,\ h[4]=13$$

- Hence

$$\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ 0 \\ 0 \end{bmatrix}=\begin{bmatrix} 2 & 0 & 0 \\ 4 & 2 & 0 \\ -5 & 4 & 2 \\ -3 & -5 & 4 \\ 13 & -3 & -5 \end{bmatrix}\begin{bmatrix} 1 \\ d_1 \\ d_2 \end{bmatrix}$$

Structure Verification

- Solving we get

$$\begin{bmatrix} d_1 \\ d_2 \end{bmatrix} = \begin{bmatrix} -5 & 4 \\ -3 & -5 \end{bmatrix}^{-1} \begin{bmatrix} -3 \\ 13 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

and

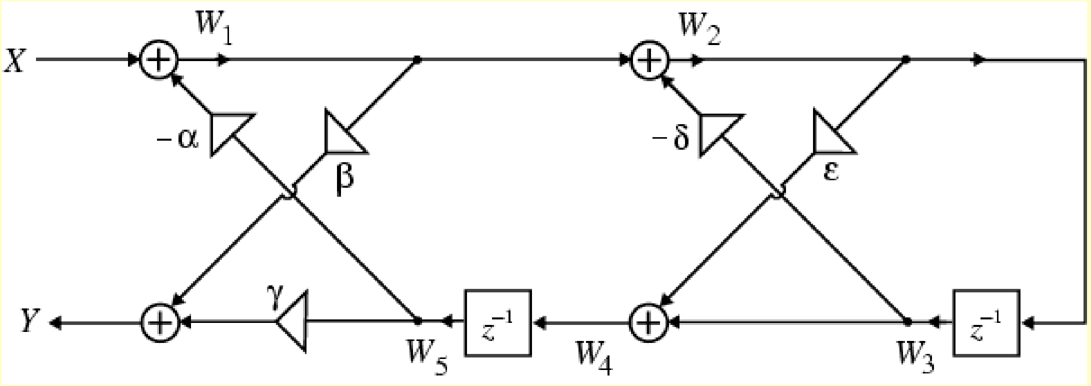
$$\begin{bmatrix} p_0 \\ p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 4 & 2 & 0 \\ -5 & 4 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 6 \\ 3 \end{bmatrix}$$

Structure Simulation and Verification Using MATLAB

- For computer simulation, the structure is described in the form of a set of equations
- These equations must be ordered properly to ensure computability
- The procedure is to express the output of each adder and filter output variable in terms of all incoming signal variables

Structure Simulation and Verification Using MATLAB

- Consider the structure



- A valid computational algorithm involving the least number of equations is

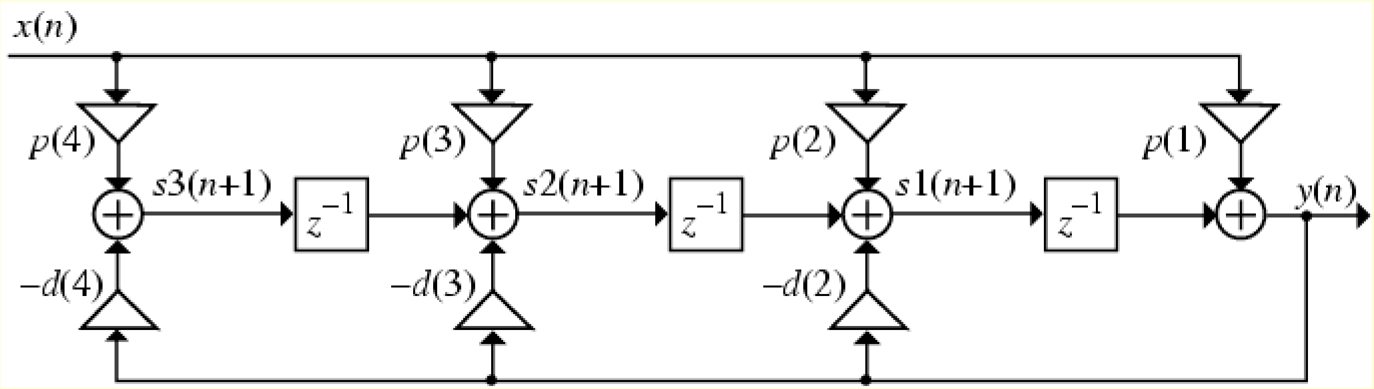
$$w_1[n] = x[n] - \alpha\, w_4[n-1],$$
$$w_2[n] = w_1[n] - \delta\, w_2[n-1],$$
$$w_4[n] = w_2[n-1] + \varepsilon\, w_2[n],$$
$$y[n] = \beta\, w_1[n] + \gamma\, w_4[n-1]$$

Structure Simulation and Verification Using MATLAB

- This set of equations is evaluated for increasing values of n starting at $n = 0$
- At the beginning, the initial conditions $w_2[-1]$ and $w_4[-1]$ can be set to any desired values, which are typically zero
- From the computed impulse response samples, the structure can be verified by determining the transfer function coefficients using the M-file `strucver`

Simulation of IIR Filters

- The M-file `filter` implements the IIR filter in the transposed direct form II structure shown below for a 3rd order filter



- As indicated in the figure, d(1) has been assumed to be equal to 1

Simulation of IIR Filters

- Basic forms of this function are

```
y = filter(num,den,x)
[y,sf]=filter(num,den,x,si)
```

  where `x` is the input vector, `y` is the output vector, `si` is the vector of initial conditions of the delay variables, and `sf` is the vector of final values of the delay variables

- For the simulation of a causal IIR filter realized in direct form II structure use the M-file `direct2`

Simulation of IIR Filters

- For the simulation of overlap-add filtering method use the M-file `fftfilt` or the second form of the M-file `filter`
- For the simulation of tapped cascaded lattice filter structures, use the M-file `latcfilt`
- The M-files `filter`, `direct2` and `latcfilt` can also be used to simulate FIR filters
- The M-file `filtfilt` implements the zero-phase filtering

Discrete Fourier Transform Computation

- The N-point DFT $X[k]$ of a length-N sequence $x[n]$, $0 \le n \le N-1$, is defined by

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \quad 0 \le k \le N-1$$

where

$$W_N = e^{-j2\pi/N}$$

- Direct computation of all N samples of $\{X[k]\}$ requires N^2 complex multiplications and $N(N-1)$ complex additions

Goertzel's Algorithm

- A recursive DFT computation scheme that makes use of the identity

$$W_N^{-kN} = 1$$

obtained using the periodicity of W_N^{-kn}

- Using this identity we can write

$$X[k] = \sum_{\ell=0}^{N-1} x[\ell] W_N^{k\ell}$$

$$= W_N^{-kN} \sum_{\ell=0}^{N-1} x[\ell] W_N^{k\ell} = \sum_{\ell=0}^{N-1} x[\ell] W_N^{-k(N-\ell)}$$

Goertzel's Algorithm

- Define $y_k[n] = \sum_{\ell=0}^{n} x_e[\ell] W_N^{-k(n-\ell)}$
- Note: $y_k[n]$ is the direct convolution of the causal sequence

$$x_e[n] = \begin{cases} x[n], & 0 \le n \le N-1 \\ 0, & n < 0,\ n \ge N \end{cases}$$

with a causal sequence

$$h_k[n] = \begin{cases} W_N^{-kn}, & n \ge 0 \\ 0, & n < 0 \end{cases}$$

- Observe $X[k] = y_k[n]\big|_{n=N}$

Goertzel's Algorithm

- z-transform of $y_k[n] = \sum_{\ell=0}^{n} x_e[\ell] W_N^{-k(n-\ell)}$ yields

$$Y_k(z) = \mathcal{Z}\{y_k[n]\} = \frac{X_e(z)}{1 - W_N^{-k} z^{-1}} = H_k(z) X_e(z)$$

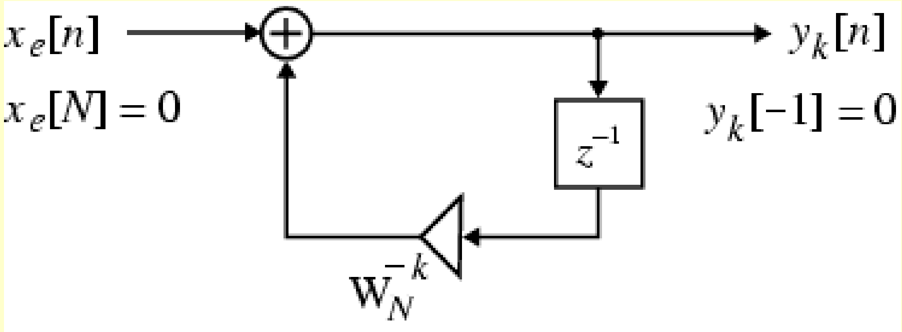
where $H_k(z) = \mathcal{Z}\{h_k[n]\} = 1/(1 - W_N^{-k} z^{-1})$

and $X_e(z) = \mathcal{Z}\{x_e[n]\}$

- Thus, $y_k[n]$ is the output of an initially relaxed LTI digital filter $H_k(z)$ with an input $x_e[n]$ and, when $n = N$, $y_k[N] = X[k]$

Goertzel's Algorithm

- Structural interpretation of the algorithm -



- Thus a recursive DFT computation scheme is

$$y_k[n] = x_e[n] + W_N^{-k} y_k[n-1], \quad 0 \le n \le N$$

with $y_k[-1] = 0$ and $x_e[N] = 0$

Goertzel's Algorithm

- Since a complex multiplication can be implemented with 4 real multiplications and 2 real additions, computation of each new value of $y_k[n]$ requires 4 real multiplications and 4 real additions
- Thus computation of $X[k] = y_k[N]$ involves $4N$ real multiplications and $4N$ real additions

→ Computation of all N DFT samples requires $4N^2$ real multiplications and $4N^2$ real additions

Goertzel's Algorithm

- Recall, direct computation of all N samples of $\{X[k]\}$ requires N^2 complex multiplications and $N(N-1)$ complex additions
- Equivalently, direct computation of all N samples of $\{X[k]\}$ requires $4N^2$ real multiplications and $N(4N-2)$ real additions
- Thus, Goertzel's algorithm requires $2N$ more real additions than the direct DFT computation

Goertzel's Algorithm

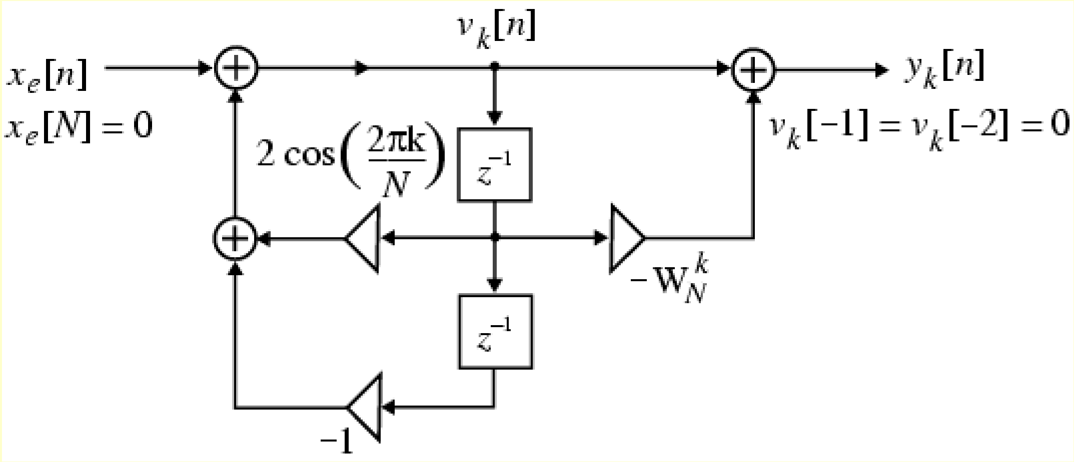
- Algorithm can be made computationally more efficient by observing that $H_k(z)$ can be rewritten as

$$H_k(z) = \frac{1}{1 - W_N^{-k} z^{-1}} = \frac{1 - W_N^k z^{-1}}{(1 - W_N^{-k} z^{-1})(1 - W_N^k z^{-1})}$$

$$= \frac{1 - W_N^k z^{-1}}{1 - 2\cos(2\pi k / N) z^{-1} + z^{-2}}$$

resulting in a second-order realization

Goertzel's Algorithm



- DFT computation equations are now

$$v_k[n] = x_e[n] + 2\cos(2\pi k/N)\, v_k[n-1] - v_k[n-2],\ 0 \le n \le N$$

$$X[k] = y_k[N] = v_k[N] - W_N^k v_k[N-1]$$

Goertzel's Algorithm

- Computation of each sample of $v_k[n]$ involves only 2 real multiplications and 4 real additions
- Complex multiplication by W_N^k needs to be performed only once at $n = N$
- Thus, computation of one sample of $X[k]$ requires $(2N+4)$ real multiplications and $(4N+4)$ real additions
- Computation of all N DFT samples requires $2N(N+2)$ real multiplications and $4N(N+1)$ real additions

Goertzel's Algorithm

- In the realization of $H_{N-k}(z)$, the multiplier in the feedback path is

$$2\cos(2\pi(N-k)/N) = 2\cos(2\pi k/N)$$

which is same as that in the realization of $H_k(z)$

$\Rightarrow$ $v_{N-k}[n] = v_k[n]$, i.e., the intermediate variables computed to determine $X[k]$ can again be used to determine $X[N-k]$

- Only difference between the two structures is the feed-forward multiplier which is now W_N^{-k}, that is the complex conjugate of W_N^k

Goertzel's Algorithm

- Thus, computation of $X[k]$ and $X[N-k]$ require $2(N+4)$ real multiplications and $4(N+2)$ real additions
- Computation of all N DFT samples require approximately N^2 real multiplications and approximately $2N^2$ real additions
- Number of real multiplications is about one-fourth and number of real additions is about one-half of those needed in direct DFT computation

Decimation-in-Time FFT Algorithm

- Consider a sequence $x[n]$ of length $N = 2^{\mu}$
- Using a 2-band polyphase decomposition we can express its z-transform as

$$X(z) = X_0(z^2) + z^{-1} X_1(z^2)$$

where

$$X_0(z) = \sum_{n=0}^{(N/2)-1} x_0[n] z^{-n} = \sum_{n=0}^{(N/2)-1} x[2n] z^{-n}$$

$$X_1(z) = \sum_{n=0}^{(N/2)-1} x_1[n] z^{-n} = \sum_{n=0}^{(N/2)-1} x[2n+1] z^{-n}$$

Decimation-in-Time FFT Algorithm

- Evaluating on the unit circle at N equally spaced points $z = W_N^{-k}, \; 0 \le k \le N-1,$ we arrive at the N-point DFT of $x[n]$:

$$X[k] = X_0[\langle k \rangle_{N/2}] + W_N^k X_1[\langle k \rangle_{N/2}],$$
$$0 \le k \le N-1$$

where $X_0[k]$ and $X_1[k]$ are the $(N/2)$-point DFTs of the $(N/2)$-length sequences $x_0[n]$ and $x_1[n]$

Decimation-in-Time FFT Algorithm

- i.e., $X_0[k] = \sum_{r=0}^{(N/2)-1} x_0[r] W_{N/2}^{rk}$

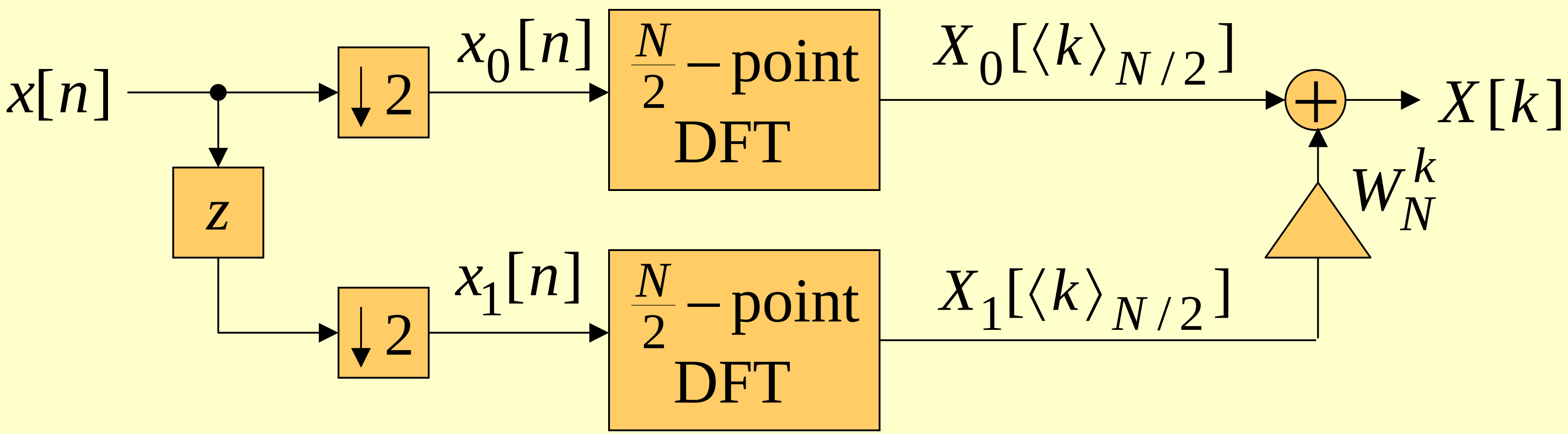
$$= \sum_{r=0}^{(N/2)-1} x[2r] W_{N/2}^{rk}, \; 0 \le k \le \frac{N}{2} - 1$$

$$X_1[k] = \sum_{r=0}^{(N/2)-1} x_1[r] W_{N/2}^{rk}$$

$$= \sum_{r=0}^{(N/2)-1} x[2r+1] W_{N/2}^{rk}, \; 0 \le k \le \frac{N}{2} - 1$$

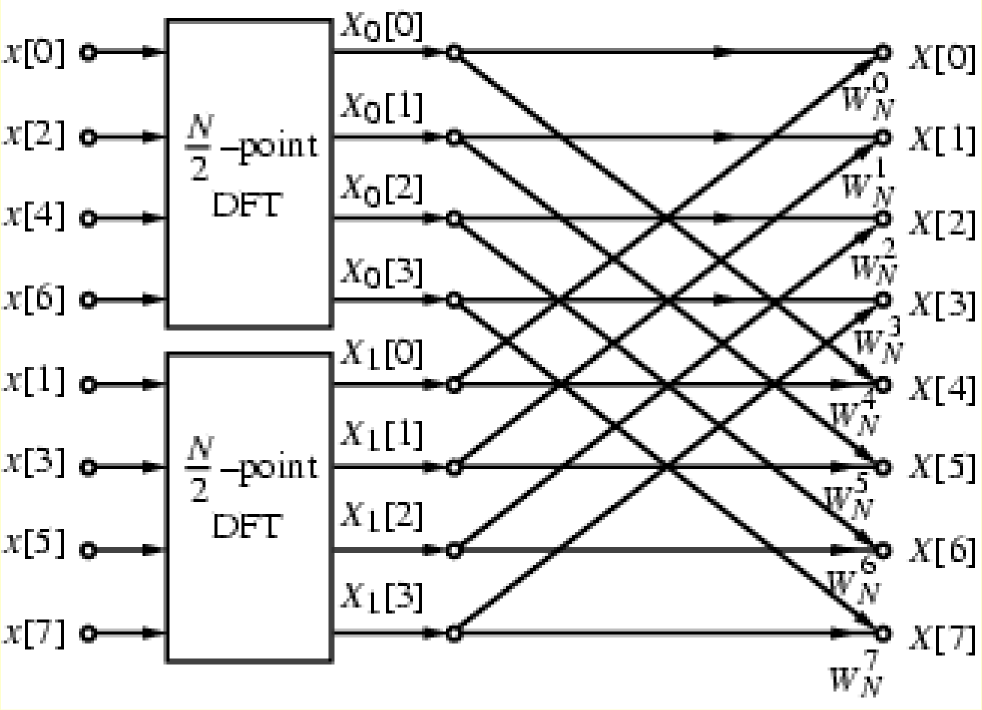
Decimation-in-Time FFT Algorithm

- Block-diagram interpretation



Decimation-in-Time FFT Algorithm

- Flow-graph representation



Decimation-in-Time FFT Algorithm

- Direct computation of the N-point DFT requires N^2 complex multiplications and $N^2 - N \approx N^2$ complex additions
- Computation of the N-point DFT using the modified scheme requires the computation of two $(N/2)$-point DFTs that are then combined with N complex multiplications and N complex additions resulting in a total of $(N^2/2) + N$ complex multiplications and approximately $(N^2/2) + N$ complex additions

Decimation-in-Time FFT Algorithm

- For $N \geq 3$, $(N^2/2) + N < N^2$
- Continuing the process we can express $X_0[k]$ and $X_1[k]$ as a weighted combination of two (N/4)-point DFTs
- For example, we can write

$$X_0[k] = X_{00}[\langle k \rangle_{N/4}] + W_{N/2}^{k} X_{01}[\langle k \rangle_{N/4}],$$
$$0 \leq k \leq (N/2) - 1$$

where $X_{00}[k]$ and $X_{01}[k]$ are the (N/4)-point DFTs of the (N/4)-length sequences $x_{00}[n] = x_0[2n]$ and $x_{01}[n] = x_0[2n+1]$

Decimation-in-Time FFT Algorithm

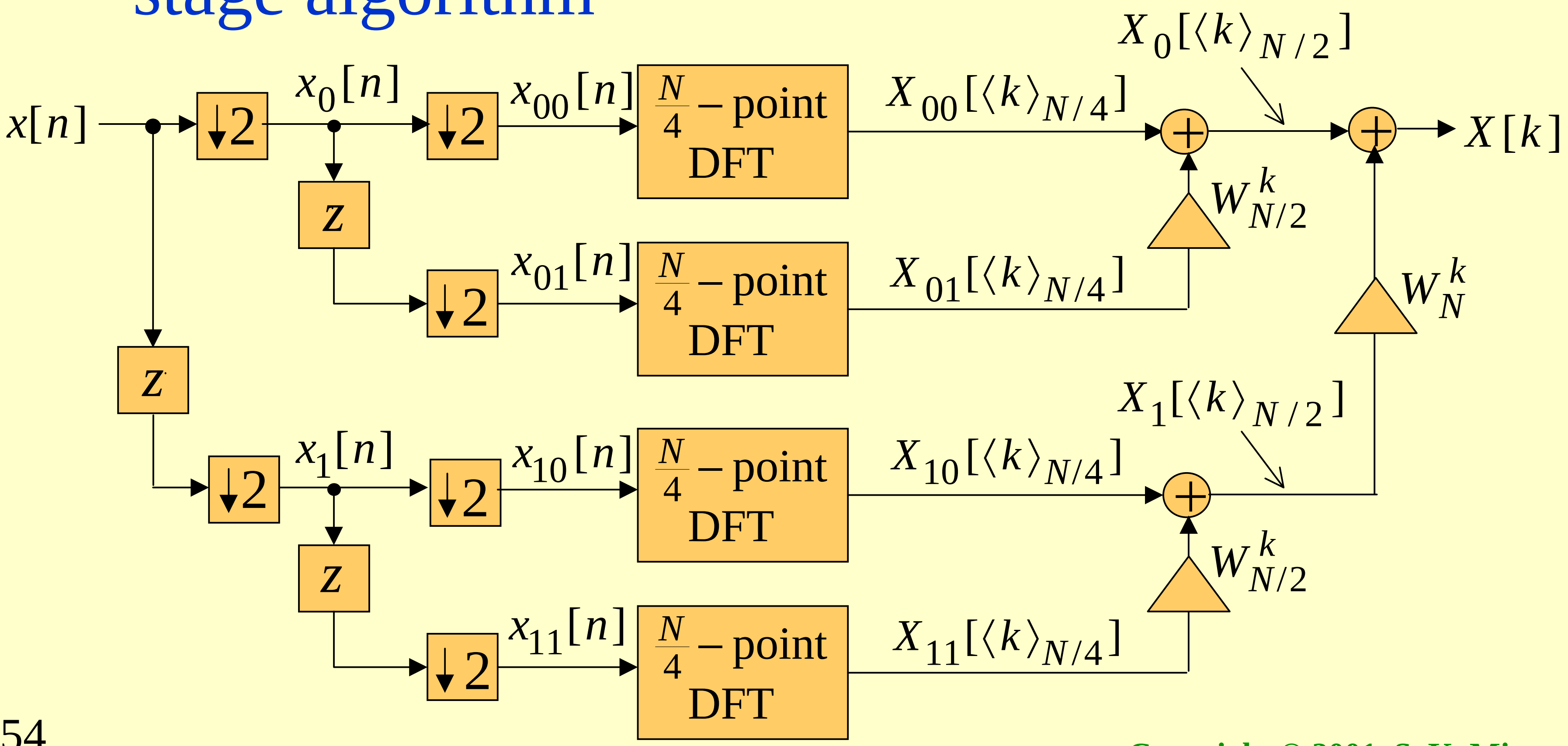
- Likewise, we can express

$$X_1[k] = X_{10}[\langle k \rangle_{N/4}] + W_{N/2}^{k} X_{11}[\langle k \rangle_{N/4}],$$
$$0 \le k \le (N/2) - 1$$

where $X_{10}[k]$ and $X_{11}[k]$ are the $(N/4)$-point DFTs of the $(N/4)$-length sequences $x_{10}[n] = x_1[2n]$ and $x_{11}[n] = x_1[2n+1]$

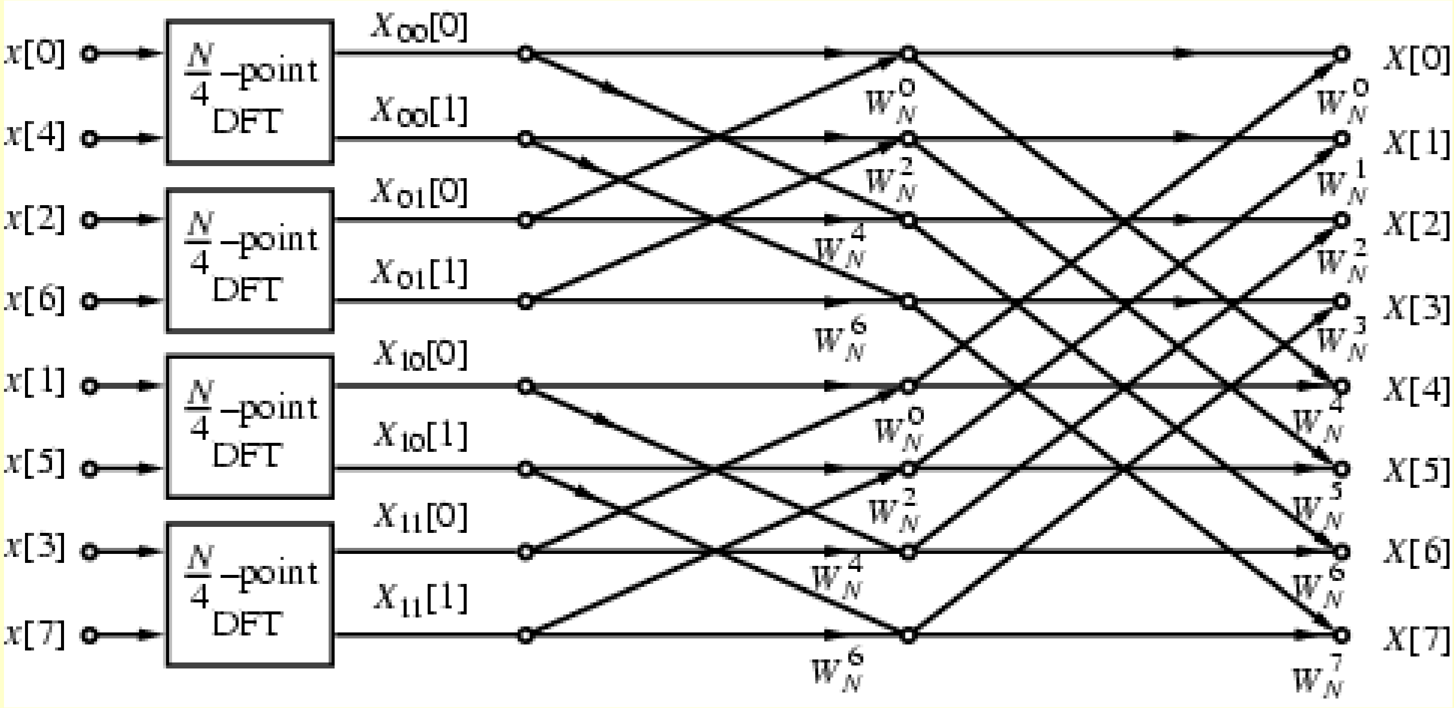
Decimation-in-Time FFT Algorithm

- Block-diagram representation of the two-stage algorithm



Decimation-in-Time FFT Algorithm

- Flow-graph representation



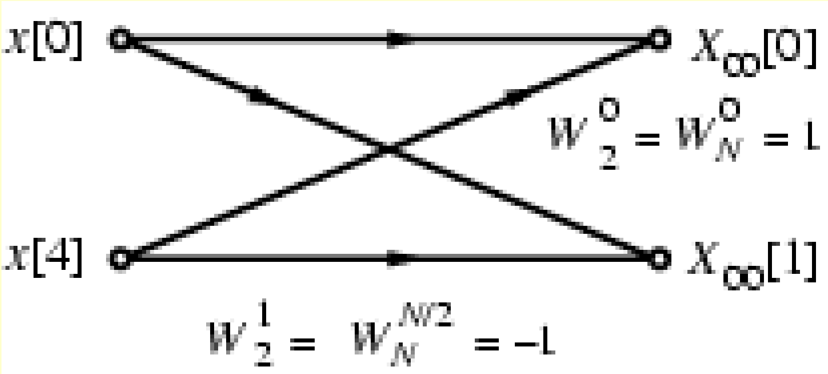
Decimation-in-Time FFT Algorithm

- In the flow-graph shown $N = 8$
- Hence, the $(N/4)$-point DFT here is a 2-point DFT and no further decomposition is possible
- The four 2-point DFTs, $X_{ij}[k], \;\; i, j = 0,1$ can be easily computed
- For example

$$X_{00}[k] = x[0] + W_2^k x[4], \;\; k = 0,1$$

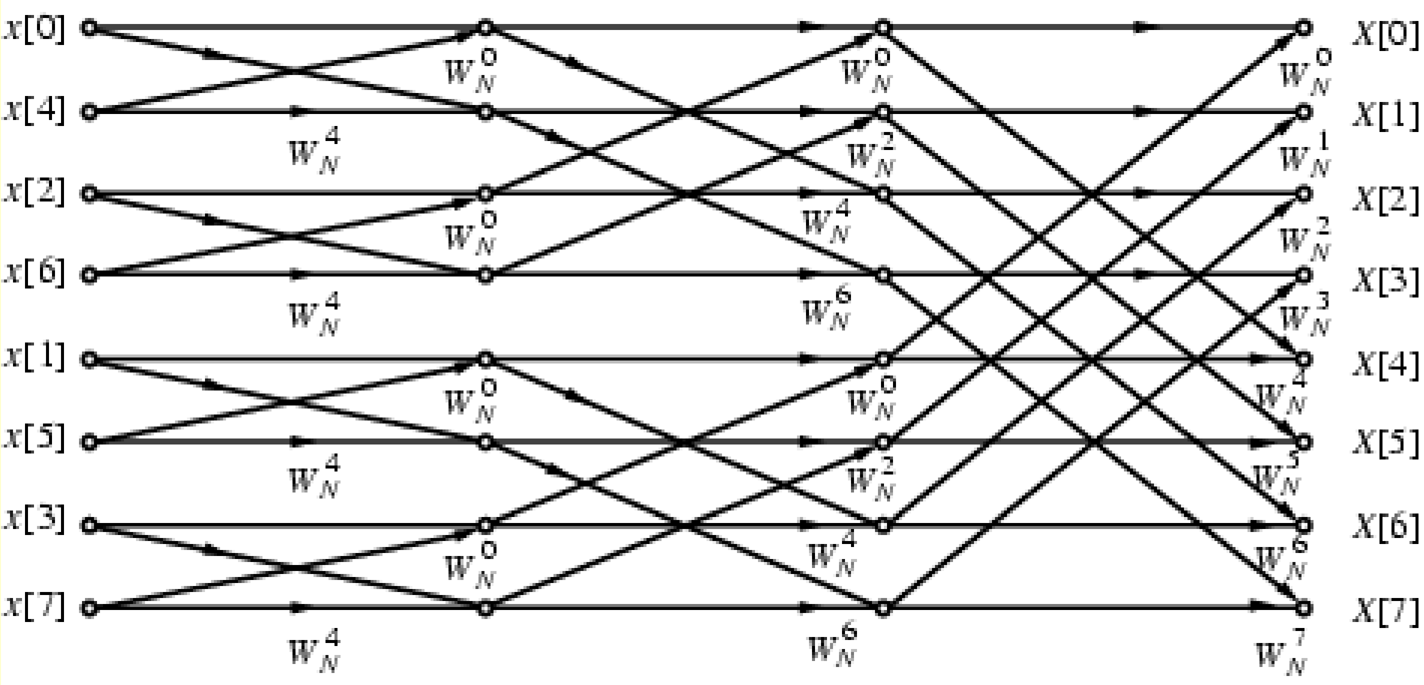
Decimation-in-Time FFT Algorithm

- Corresponding flow-graph of the 2-point DFT is shown below obtained using the identity $W_2^k = W_N^{(N/2)k}$



Decimation-in-Time FFT Algorithm

- Complete flow-graph of the 8-point DFT is shown below



Decimation-in-Time FFT Algorithm

- The flow-graph consists of 3 stages
- First stage computes the **four** 2-point DFTs
- Second stage computes the **two** 4-point DFTs
- Last stage computes the desired 8-point DFT
- The number of complex multiplications and additions at each stage is equal to 8, the size of the DFT

Decimation-in-Time FFT Algorithm

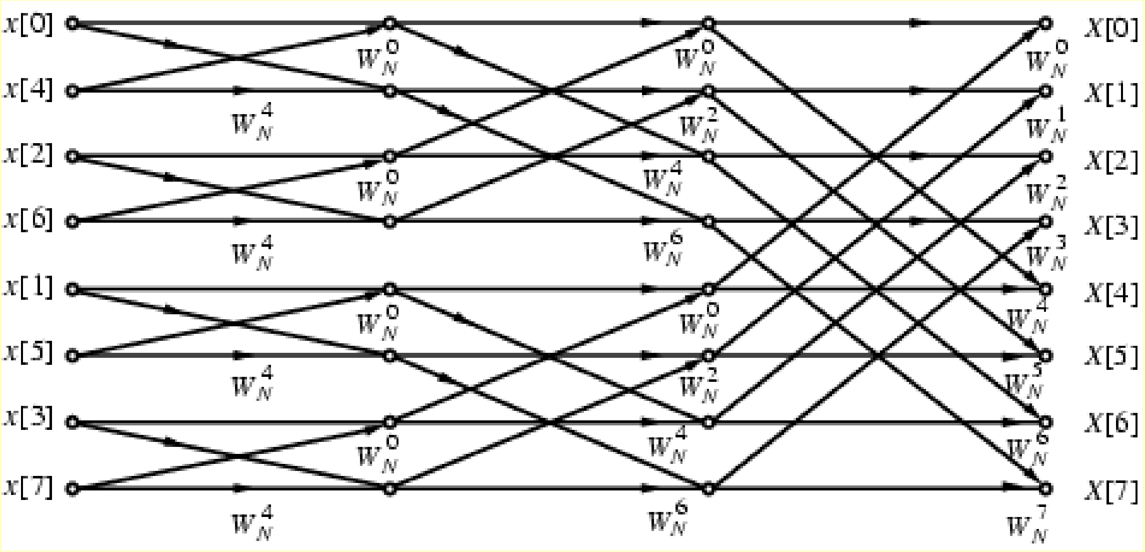
- Total number of complex multiplications and additions to compute all 8 DFT samples is equal to $8 + 8 + 8 = 24 = 8 \times 3$
- In the general case when $N = 2^{\mu}$, number of stages for the computation of the (2^{μ})-point DFT in the fast algorithm will be $\mu = \log_2 N$
- Total number of complex multiplications and additions to compute all N DFT samples is $N(\log_2 N)$

Decimation-in-Time FFT Algorithm

- In developing the count, multiplications with $W_N^0 = 1$ and $W_N^{N/2} = -1$ have been assumed to be complex
- Also the symmetry property of
$$W_N^{(N/2)+k} = -W_N^k$$
has not been taken advantage of
- These properties can be exploited to reduce the computational complexity further

Decimation-in-Time FFT Algorithm

- Examination of the flow-graph

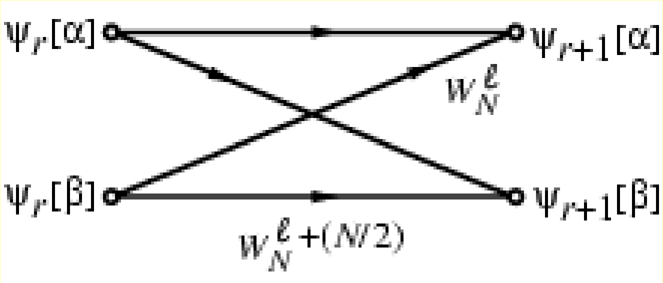


reveals that each stage of the DFT computation process employs the same basic computational module

Decimation-in-Time FFT Algorithm

- In the basic module two output variables are generated by a weighted combination of two input variables as indicated below

  where $r = 1,2,\ldots,\mu$ and $\alpha, \beta = 0,1,\ldots,N-1$



- Basic computational module is called a **butterfly computation**

Decimation-in-Time FFT Algorithm

- Input-output relations of the basic module are:

$$\Psi_{r+1}[\alpha] = \Psi_r[\alpha] + W_N^{\ell}\Psi_r[\beta]$$

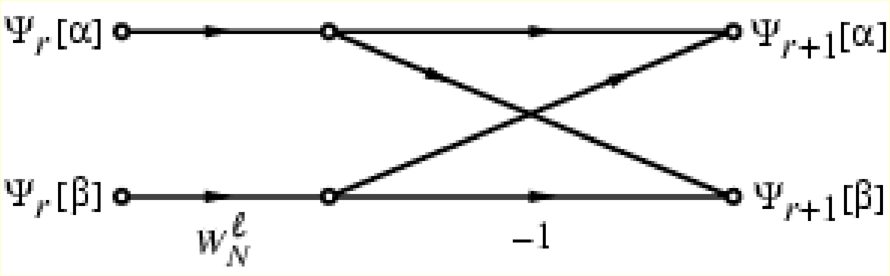
$$\Psi_{r+1}[\beta] = \Psi_r[\alpha] + W_N^{\ell+(N/2)}\Psi_r[\beta]$$

- Substituting $W_N^{\ell+(N/2)} = -W_N^{\ell}$ in the second equation given above we get

$$\Psi_{r+1}[\beta] = \Psi_r[\alpha] - W_N^{\ell}\Psi_r[\beta]$$

Decimation-in-Time FFT Algorithm

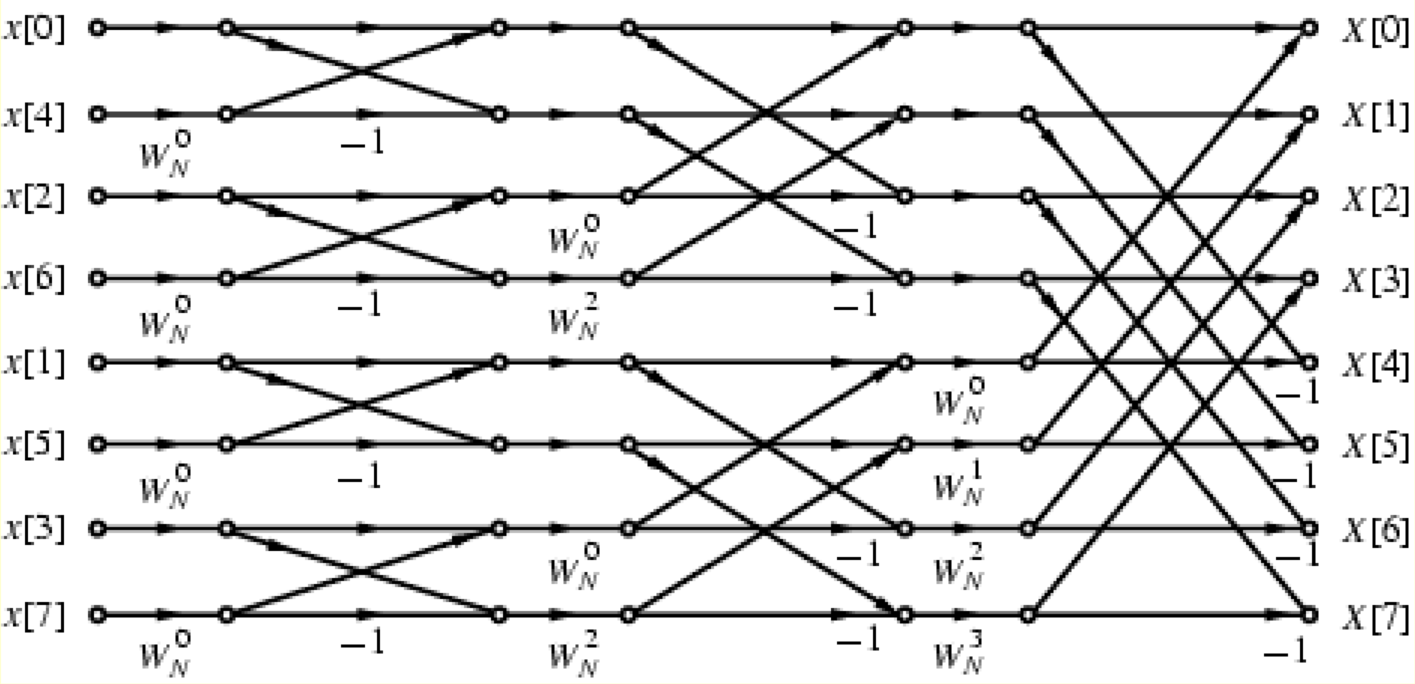
- Modified butterfly computation requires only one complex multiplication as indicated below



- Use of the above modified butterfly computation module reduces the total number of complex multiplications by 50%

Decimation-in-Time FFT Algorithm

- New flow-graph using the modified butterfly computational module for $N = 8$



Decimation-in-Time FFT Algorithm

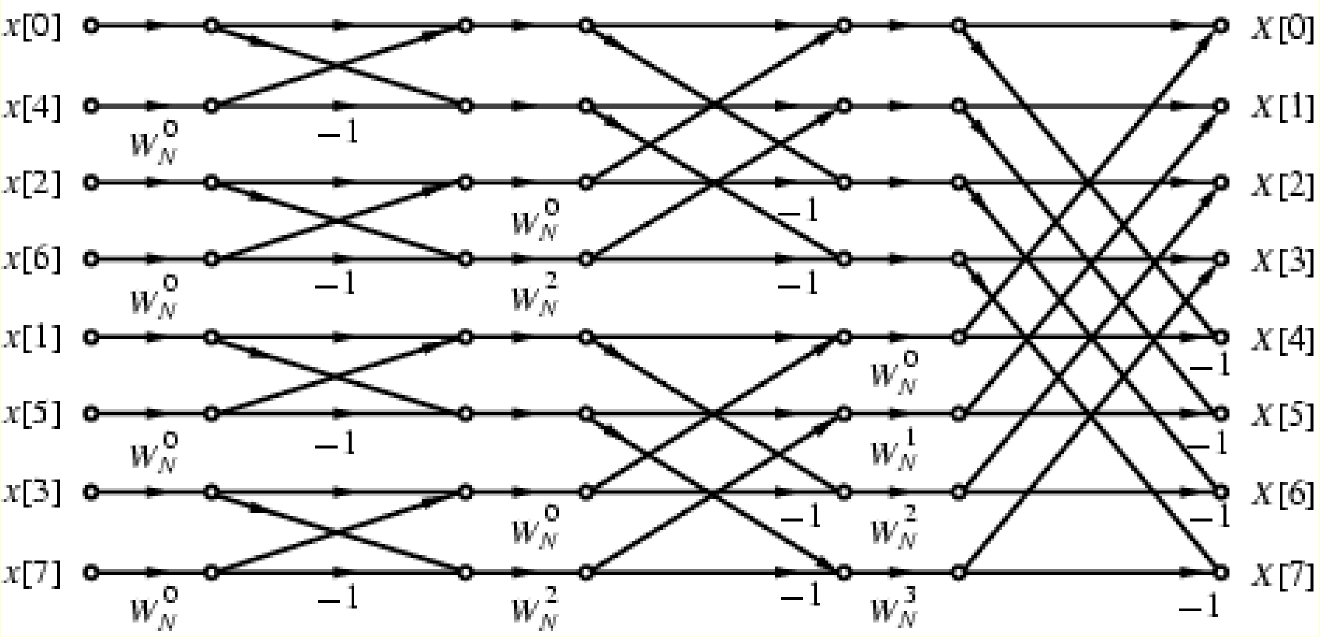
- Computational complexity can be reduced further by avoiding multiplications by $W_N^0 = 1$, $W_N^{N/2} = -1$, $W_N^{N/4} = j$, and $W_N^{3N/4} = -j$
- The DFT computation algorithm described here also is efficient with regard to memory requirements
- Note: Each stage employs the same butterfly computation to compute $\Psi_{r+1}[\alpha]$ and $\Psi_{r+1}[\beta]$ from $\Psi_r[\alpha]$ and $\Psi_r[\beta]$

Decimation-in-Time FFT Algorithm

- At the end of computation at any stage, output variables $\Psi_{r+1}[m]$ can be stored in the same registers previously occupied by the corresponding input variables $\Psi_r[m]$
- This type of memory location sharing is called **in-place computation** resulting in significant savings in overall memory requirements

Decimation-in-Time FFT Algorithm

- In the DFT computation scheme outlined, the DFT samples $X[k]$ appear at the output in a sequential order while the input samples $x[n]$ appear in a different order



Decimation-in-Time FFT Algorithm

- Thus, a sequentially ordered input $x[n]$ must be reordered appropriately before the fast algorithm described by this structure can be implemented
- To understand the input reordering scheme represent the arguments of input samples x[n] and their sequentially ordered new representations $\Psi_1[m]$ in binary forms

Decimation-in-Time FFT Algorithm

- The relations between the arguments m and n are as follows:

$$\begin{array}{lllllllll} m: & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\ n: & 000 & 100 & 010 & 110 & 001 & 101 & 011 & 111 \end{array}$$

- Thus, if $(b_2 b_1 b_0)$ represents the index n of $x[n]$, then the sample $x[b_2 b_1 b_0]$ appears at the location $m = b_0 b_1 b_2$ as $\Psi_1[b_0 b_1 b_2]$ before the DFT computation is started
- i.e., location of $\Psi_1[m]$ is in **bit-reversed order** from that of $x[n]$

Decimation-in-Time FFT Algorithm

- Alternative forms of the fast DFT algorithms can be obtained by reordering the computations such as input in normal order and output in bit-reversed order, and both input and output in normal order
- The fast algorithm described assumes that the length of $x[n]$ is a power of 2
- If it is not, the length can be extended by zero-padding and make the length a power of 2

Decimation-in-Time FFT Algorithm

- Even after zero-padding, the DFT computation based on the fast algorithm may be computationally more efficient than a direct DFT computation of the original shorter sequence
- The fast DFT computation schemes described are called **decimation-in-time** (DIT) **fast Fourier transform** (FFT) algorithms as input $x[n]$ is first decimated to form a set of subsequences before the DFT is computed

Decimation-in-Time FFT Algorithm

- For example, the relation between $x[n]$ and its even and odd parts, $x_0[n]$ and $x_1[n]$, generated by the first stage of the DIT algorithm is given by

$x[n]$:	$x[0]$	$x[1]$	$x[2]$	$x[3]$	$x[4]$	$x[5]$	$x[6]$	$x[7]$
$x_0[n]$:	$x[0]$		$x[2]$		$x[4]$		$x[6]$	
$x_1[n]$:	$x[1]$		$x[3]$		$x[5]$		$x[7]$	

Decimation-in-Time FFT Algorithm

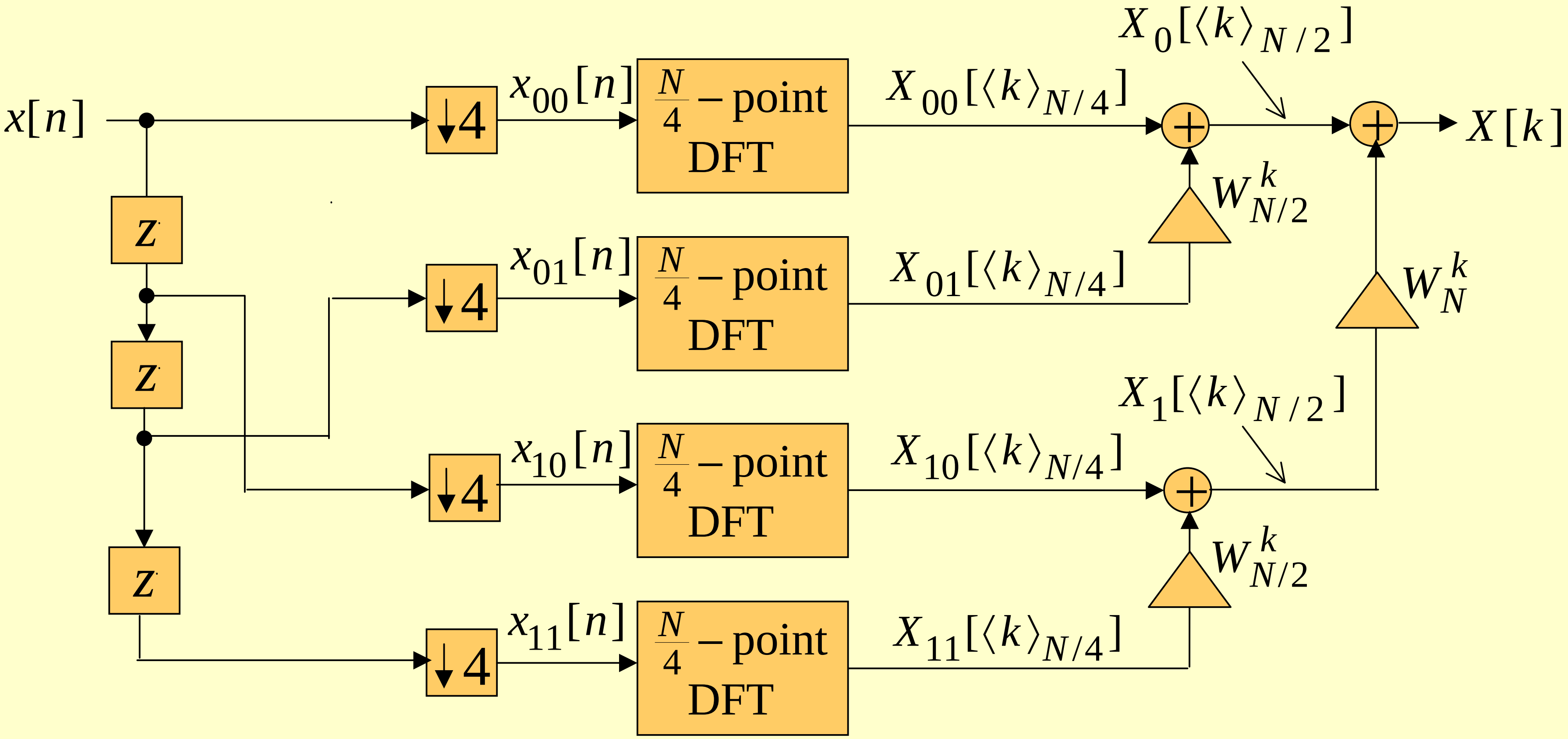
- Likewise, the relation between $x[n]$ and the sequences $x_{00}[n]$, $x_{01}[n]$, $x_{10}[n]$, and $x_{11}[n]$, generated by the two-stage decomposition of the DIT algorithm is given by

$x[n]$:	$x[0]$	$x[1]$	$x[2]$	$x[3]$	$x[4]$	$x[5]$	$x[6]$	$x[7]$
$x_{00}[n]$:	$x[0]$				$x[4]$			
$x_{01}[n]$:	$x[2]$				$x[6]$			
$x_{10}[n]$:	$x[1]$				$x[5]$			
$x_{11}[n]$:	$x[3]$				$x[7]$			

Decimation-in-Time FFT Algorithm

- The subsequences $x_{00}[n]$, $x_{01}[n]$, $x_{10}[n]$, and $x_{11}[n]$ can be generated directly by a factor-of-4 decimation process leading to a single-stage decomposition as shown on the next slide

Decimation-in-Time FFT Algorithm



Decimation-in-Time FFT Algorithm

- **Radix-*R* FFT algorithm -** A each stage the decimation is by a factor of R
- Depending on N, various combinations of decompositions of $X[k]$ can be used to develop different types of DIT FFT algorithms
- If the scheme uses a mixture of decimations by different factors, it is called a **mixed radix FFT algorithm**

Decimation-in-Time FFT Algorithm

- For N which is a composite number expressible in the form of a product of integers:

$$N = r_1 \cdot r_2 \cdots r_\nu$$

total number of complex multiplications (additions) in a DIT FFT algorithm based on a ν-stage decomposition is given by

$$\left(\sum_{i=1}^{\nu} r_i - \nu \right) N$$

Decimation-in-Frequency FFT Algorithm

- Consider a sequence $x[n]$ of length $N = 2^{\mu}$
- Its z-transform can be expressed as

$$X(z) = X_a(z) + z^{-N/2} X_b(z)$$

where

$$X_a(z) = \sum_{n=0}^{(N/2)-1} x[n] z^{-n}$$

$$X_b(z) = \sum_{n=0}^{(N/2)-1} x[\tfrac{N}{2} + n] z^{-n}$$

Decimation-in-Frequency FFT Algorithm

- Evaluating $X(z)$ on the unit circle at we get

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

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

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

Decimation-in-Frequency FFT Algorithm

- For k even

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

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

- For k odd

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

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

Decimation-in-Frequency FFT Algorithm

- We can write

$$X[2\ell]=\sum_{n=0}^{(N/2)-1} x_0[n]\,W_N^{n(2\ell)}$$

$$X[2\ell+1]=\sum_{n=0}^{(N/2)-1} x_1[n]\,W_N^{n(2\ell)},\ \ 0\le \ell \le \frac{N}{2}-1$$

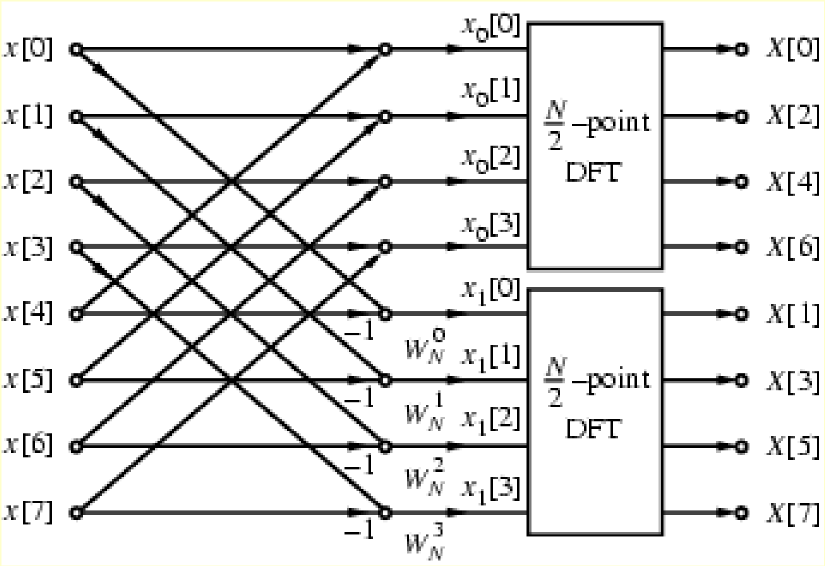
where

$$x_0[n]=(x[n]+x[\tfrac{N}{2}+n]),$$

$$x_1[n]=(x[n]-x[\tfrac{N}{2}+n])W_N^{n},\quad 0\le n\le \frac{N}{2}-1$$

Decimation-in-Frequency FFT Algorithm

- Thus $X[2\ell]$ and $X[2\ell+1]$ are the $(N/2)$-point DFTs of the length-$(N/2)$ sequences $x_0[n]$ and $x_1[n]$
- Flow-graph of the first-stage of the DFT algorithm is shown below



Decimation-in-Frequency FFT Algorithm

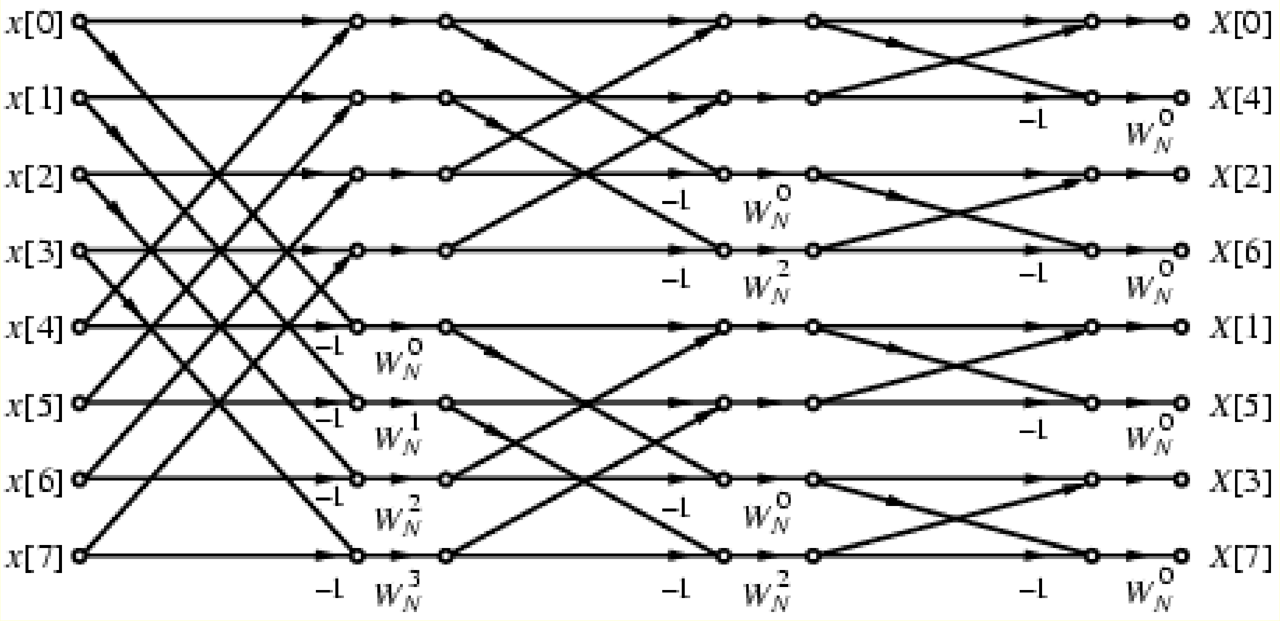
- Here the input samples are in sequential order, while the output DFT samples appear in a decimated form with the even-indexed samples appearing as the output of one $(N/2)$-point DFT and the odd-indexed samples appearing as the output of the other $(N/2)$-point DFT

Decimation-in-Frequency FFT Algorithm

- We next express the even- and odd-indexed samples of each one of the two $(N/2)$-point DFTs as a sum of two $(N/4)$-point DFTs
- Process is continued until the smallest DFTs are 2-point DFTs

Decimation-in-Frequency FFT Algorithm

- Complete flow-graph of the decimation-in-frequency FFT computation scheme for $N = 8$



Decimation-in-Frequency FFT Algorithm

- Computational complexity of the radix-2 DIF FFT algorithm is same as that of the DIT FFT algorithm
- Various forms of DIF FFT algorithm can similarly be developed
- The DIT and DIF FFT algorithms described here are often referred to as the **Cooley-Tukey FFT algorithms**

Inverse DFT Computation

- An FFT algorithm for computing the DFT samples can also be used to calculate efficiently the inverse DFT (IDFT)
- Consider a length-N sequence $x[n]$ with an N-point DFT $X[k]$
- Recall

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k] W_N^{-nk}$$

Inverse DFT Computation

- Multiplying both sides by N and taking the complex conjugate we get

$$Nx^*[n] = \sum_{k=0}^{N-1} X^*[k] W_N^{nk}$$

- Right-hand side of above is the N-point DFT of a sequence $X^*[k]$

Inverse DFT Computation

- Desired IDFT $x[n]$ is then obtained as

$$x[n] = \frac{1}{N}\left\{\sum_{k=0}^{N-1} X^*[k] W_N^{nk}\right\}^*$$

- Inverse DFT computation is shown below: