Computation of DFT

- Efficient *algorithms* for computing DFT – Fast Fourier Transform.

  (a) Compute only a few points out of all *N* points

  (b) Compute all *N* points

- What are the efficiency criteria?
  - Number of multiplications
  - Number of additions
  - Chip area in VLSI implementation

✧ DFT as a Linear Transformation

- Matrix representation of DFT

  Definition of DFT:

  $$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \qquad k = 0,1,\ldots,N-1$$

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

  where

  Let $\mathbf{x}_N = \begin{bmatrix} x(0) \\ x(1) \\ \vdots \\ x(N-1) \end{bmatrix}, \quad \mathbf{X}_N = \begin{bmatrix} X(0) \\ X(1) \\ \vdots \\ X(N-1) \end{bmatrix},$

  and

  $$\mathbf{W}_N = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N & W_N^2 & \cdots & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & \cdots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & W_N^{(N-1)} & W_N^{2(N-1)} & \cdots & W_N^{(N-1)(N-1)} \end{bmatrix}$$

  Thus,

  $$\mathbf{X}_N = \mathbf{W}_N \mathbf{x}_N \qquad N \text{ - point DFT}$$

  $$\mathbf{x}_N = \mathbf{W}_N^{-1} \mathbf{X}_N \qquad N \text{ - point IDFT}$$

  $$= \frac{1}{N} \mathbf{W}_N^* \mathbf{X}_N$$

  Because the matrix (transformation) $\mathbf{W}_N$ has a specific structure and because W_N^k has par-

  ticular values (for some *k* and *n*), we can reduce the number of arithmetic operations for computing this transform.

Example $x[n] = [0\ 1\ 2\ 3]$

$$\mathbf{W}_4 = \begin{bmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^1 & W_4^2 & W_4^3 \\ W_4^0 & W_4^2 & W_4^4 & W_4^6 \\ W_4^0 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_4^1 & W_4^2 & W_4^3 \\ 1 & W_4^2 & W_4^0 & W_4^2 \\ 1 & W_4^3 & W_4^2 & W_4^1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

Only additions are needed to compute this specific transform.
(This is a well-known *radix-4 FFT*)

Thus, the DFT of $x[n]$ is $\mathbf{X}_4 = \mathbf{W}_4 \mathbf{x}_4 = \begin{bmatrix} 6 \\ -2+2j \\ -2 \\ -2-2j \end{bmatrix}$

✧ **Fast Fourier Transform**

-- Highly efficient algorithms for computing DFT

- General principle: *Divide-and-conquer*
- Specific properties of W_N^k

  ■ Complex conjugate symmetry: $W_N^{-kn} = (W_N^{kn})^*$

  ■ Symmetry: $W_N^{k+N/2} = -W_N^k$

  ■ Periodicity: $W_N^{k+N} = W_N^k$

  ■ Particular values of k and n: e.g., radix-4 FFT (no multiplications)

- Direct computation of DFT

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

$$= \sum_{n=0}^{N-1} \left\{ \begin{array}{l} \left[\mathrm{Re}(x[n]) \cdot \mathrm{Re}(W_N^{kn}) - \mathrm{Im}(x[n]) \cdot \mathrm{Im}(W_N^{kn}) \right] + \\ j\left[\mathrm{Re}(x[n]) \cdot \mathrm{Im}(W_N^{kn}) + \mathrm{Im}(x[n]) \mathrm{Re}(W_N^{kn}) \right] \end{array} \right\}$$

For each k, we need N complex multiplications and N-1 complex additions. → $4N$ real multiplications and $4N$-2 real additions.

We will show how to use the properties of W_N^k to reduce computations.

- Radix-2 algorithms: Decimation-in-time; Decimation-in-frequency
- Composite *N* algorithms: Cooley-Tukey; Prime factor
- Winograd algorithm
- Chirp transform algorithm

✧ Radix-2 Decimation-in-time Algorithms

-- Assume *N*-point DFT and $N = 2^v$

- Idea: *N*-point DFT $\rightarrow$ $N/2$-point DFT $\rightarrow$ $N/4$-point DFT

$$N/4\text{-point DFT}$$

$$N/2\text{-point DFT} \rightarrow N/4\text{-point DFT}$$

$$N/4\text{-point DFT}$$

- Sequence: $x[0]\ \ x[1]\ \ x[2]\ \ x[3]\ \ \cdots\ \ x[n/2]\ \cdots\ \ x[N-1]$

  Even index: $x[0]\ \ x[2]\ \ \ \ \cdots\ \ \ x[N-2]$
  Odd index: $x[1]\ \ x[3]\ \ \ \ \cdots\ \ \ x[N-1]$

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

$$= \underbrace{\sum_{n\ \text{even}} x[n] W_N^{kn}}_{n=2r} + \underbrace{\sum_{n\ \text{odd}} x[n] W_N^{kn}}_{n=2r+1}$$

$$= \sum_{r=0}^{\frac{N}{2}-1} x[2r] W_N^{2rk} + \sum_{r=0}^{\frac{N}{2}-1} x[2r+1] W_N^{(2r+1)k}$$

$$\because W_N^2 = e^{-2j\left(\frac{2\pi}{N}\right)} = e^{-2j\left(\frac{\pi}{N/2}\right)} = W_{N/2}$$

$$X[k] = \underbrace{\sum_{r=0}^{\frac{N}{2}-1} x[2r] W_{N/2}^{rk}}_{\frac{N}{2}\text{-point DFT}} + W_N^k \underbrace{\sum_{r=0}^{\frac{N}{2}-1} x[2r+1] W_{N/2}^{rk}}_{\frac{N}{2}\text{-point DFT}}$$

$$= G[k] + W_N^k H[k]$$

- Comparison:

    (a) Direct computation of N-point DFT (N frequency samples):

    ~ N^2 complex multiplications and N^2 complex adds

    (b) Direct computation of $N/2$-point DFT:

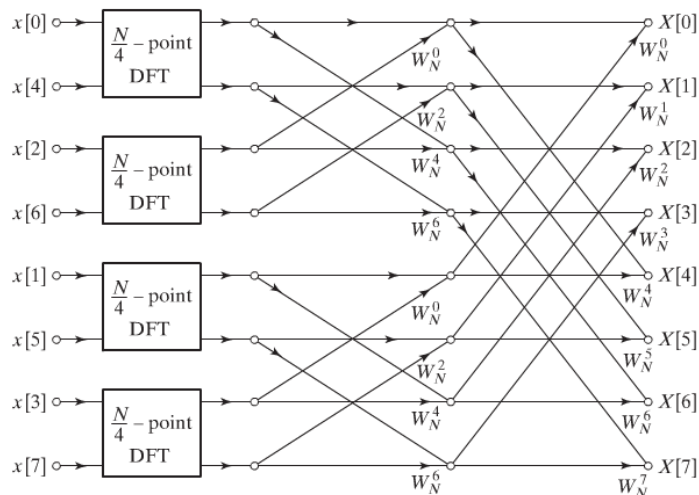   ~ $\left(\dfrac{N}{2}\right)^2$ complex multiplications and $\left(\dfrac{N}{2}\right)^2$ complex adds
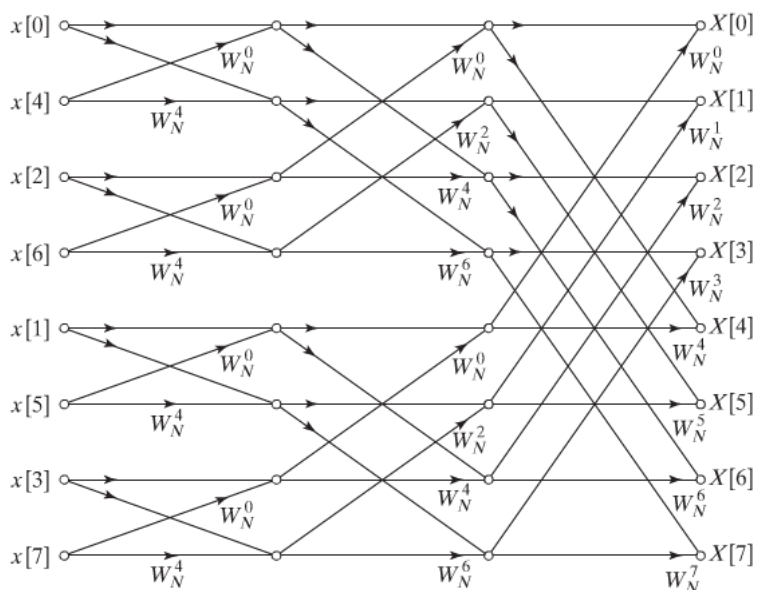
    + additional N complex multis and N complex adds

    ~ (Total:) $N + 2\left(\dfrac{N}{2}\right)^2 = N + \dfrac{N^2}{2}$ complex multis and adds

    (c) $\log_2 N$ -stage FFT

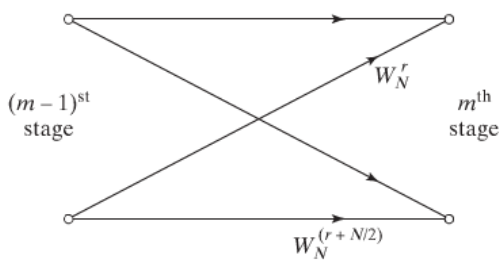   Since $N = 2^v$, we can further break $N/2$-point DFT into two $N/4$-point DFT and

    so on.

At each stage: $\sim N$ complex multis and adds

Total: $\sim N \log_2 N$ complex multis and adds $(\text{-->} \dfrac{N}{2} \log_2 N)$
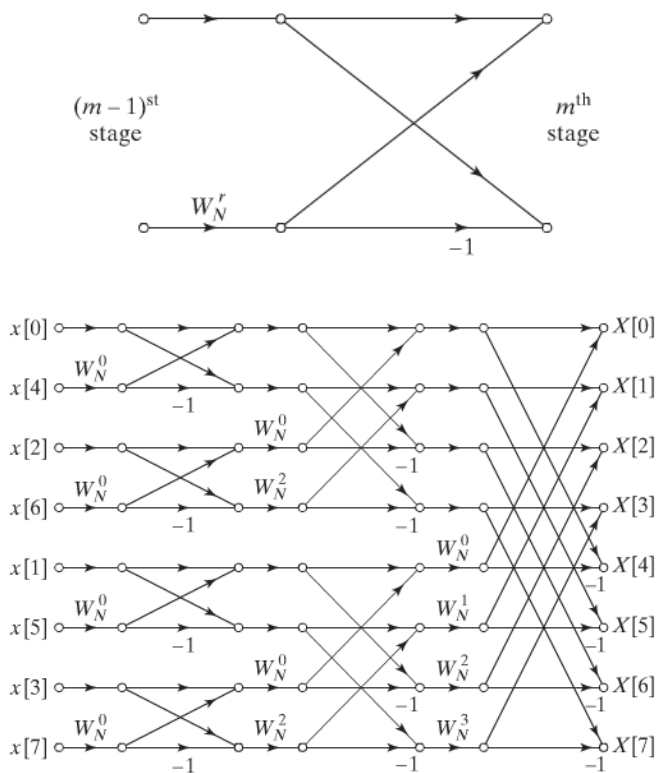
Number of points, N	Direct Computation: Complex Multis	FFT: Complex Multis	Speed Improvement Factor
4	16	4	4.0
8	64	12	5.3
16	256	32	8
64	4,096	192	21.3
256	65,536	1,024	64.0
1024	1,048,576	5,120	204.8

■ **Butterfly**: Basic unit in FFT

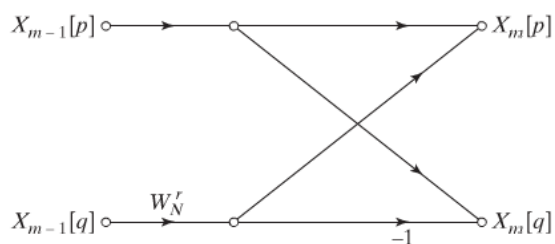Two multiplications:

One multiplication:

■ **In-place** computations

Only two registers are needed for computing a butterfly unit.

$$X_m[p] = X_{m-1}[p] + W_N^r X_{m-1}[q]$$

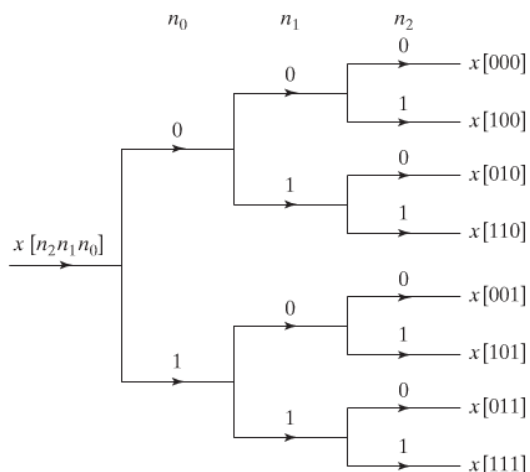$$X_m[q] = X_{m-1}[p] - W_N^r X_{m-1}[q]$$

Advantage: less storage!

■ In order to retain the in-place computation property, the input data are accessed in the **bit-reversed** order.
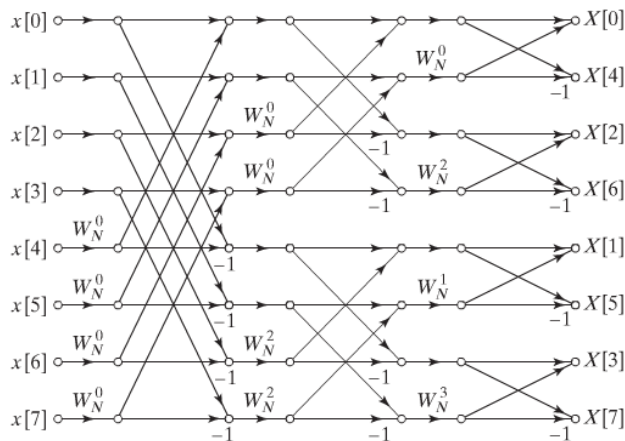
Note: The outputs are in the normal order (same as the "position")

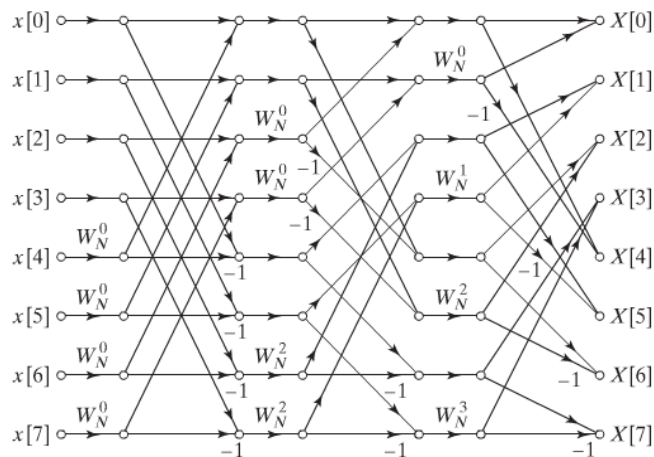| Position	Binary equivalent	Bit reversed	Sequence index
6	110	011	3
2	010	010	2

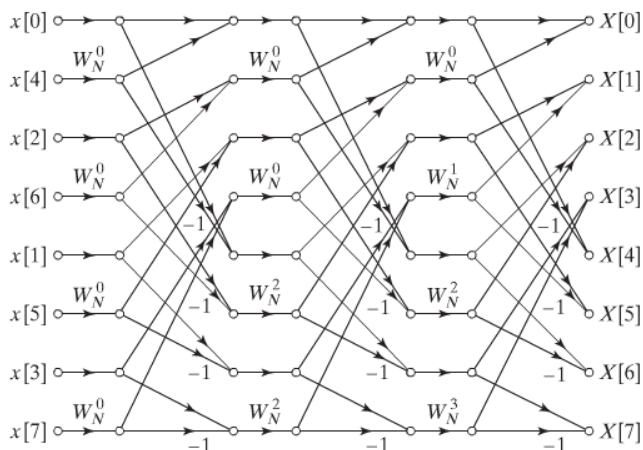*Remark:* Index 3 input data is placed at position 6.

We may also place the inputs in the normal order; then the outputs are in the bit-reversed order.

■ If we try to maintain the normal order of both inputs and outputs, then in-place computation structure is destroyed.

✧ **Radix-2 Decimation-in-frequency Algorithms**

■ Dividing the output sequence $X[k]$ into smaller pieces.

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \quad k = 0,1,\ldots,N-1$$
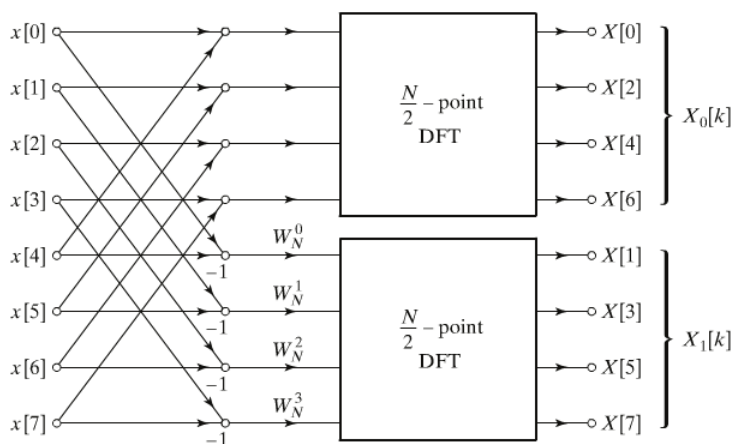
If *k* is even, $k = 2r$.

$$X[2r] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \qquad r = 0,1,\cdots,\frac{N}{2}-1$$

$$= \sum_{n=0}^{\frac{N}{2}-1} x[n] W_N^{2nr} + \sum_{n=\frac{N}{2}}^{N-1} x[n] W_N^{2nr} \qquad n \leftarrow (n + N/2)$$

$$= \sum_{n=0}^{\frac{N}{2}-1} x[n] W_N^{2nr} + \sum_{n=0}^{\frac{N}{2}-1} x\left[n + \frac{N}{2}\right] \cdot W_N^{2r\left(n+\frac{N}{2}\right)}$$

$$\because W_N^{2r[n+N/2]} = W_N^{2rm} W_N^{rN} = W_N^{2rn}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} \left(x[n] + x\left[n + \frac{N}{2}\right] \right) \cdot W_N^{2nr}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} \left(x[n] + x\left[n + \frac{N}{2}\right] \right) \cdot W_{N/2}^{nr}$$

Similarly, if *k* is odd, $k = 2r+1$.

$$X[2r+1] = \sum_{n=0}^{\frac{N}{2}-1} \left(x[n] - x\left[n + \frac{N}{2}\right] \right) \cdot W_N^{n} \cdot W_{N/2}^{nr}$$

$$\begin{cases} X[2r] = \displaystyle\sum_{n=0}^{\frac{N}{2}-1} \left(x[n] + x\left[n + \frac{N}{2}\right] \right) \cdot W_{N/2}^{nr} \\ X[2r+1] = \displaystyle\sum_{n=0}^{\frac{N}{2}-1} \left(x[n] - x\left[n + \frac{N}{2}\right] \right) \cdot W_N^n \cdot W_{N/2}^{nr} \end{cases}$$
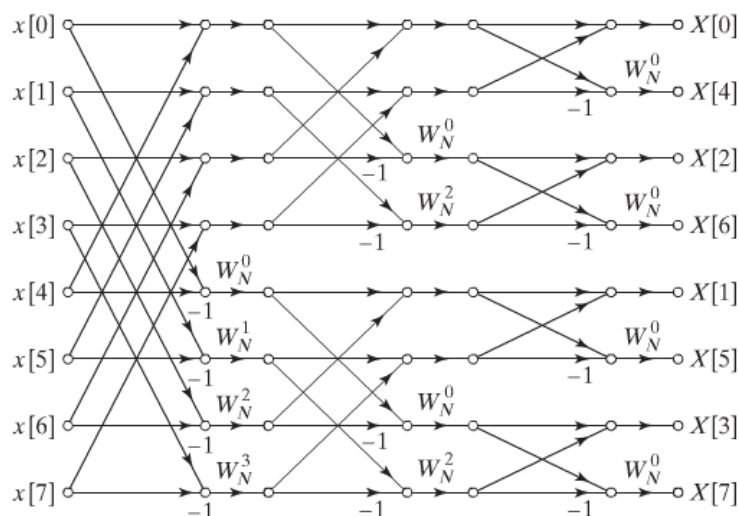
Let $\begin{cases} g[n] = x[n] + x\left[n + \dfrac{N}{2}\right] \\ h[n] = x[n] - x\left[n + \dfrac{N}{2}\right] \end{cases}$

We can further break $X[2r]$ into even and odd groups …

Again, we can reduce the two-multiplication butterfly into one multiplication. Hence, the computational complexity is bout $\dfrac{N}{2}\log_2 N$. The in-place computation property holds if the outputs are in bit-reversed order (when inputs are in the normal order).

✧ **FFT for Composite N**

-- Cooley-Tukey Algorithm: $N = N_1 N_2$

$$\begin{cases} \text{Time index}: \quad n = N_2 n_1 + n_2 & \begin{cases} 0 \le n_1 \le N_1 - 1 \\ 0 \le n_2 \le N_2 - 1 \end{cases} \\[2em] \text{Freq. index}: \quad k = k_1 + N_1 k_2 & \begin{cases} 0 \le k_1 \le N_1 - 1 \\ 0 \le k_2 \le N_2 - 1 \end{cases} \end{cases}$$

Remark: $n \leftrightarrow (n_1, n_2)$ and $k \leftrightarrow (k_1, k_2)$

■ **Goal:** Decompose N-point DFT into two stages:

N_1-point DFT $\otimes$ N_2-point DFT

$$\begin{aligned} X[k] &= \sum_{n=0}^{N-1} x[n] W_N^{kn}, \qquad 0 \le k \le N-1 \\ &= X[k_1 + N_1 k_2] \\ &= \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x[N_2 n_1 + n_2] \cdot W_N^{(k_1 + N_1 k_2)(N_2 n_1 + n_2)} \\ &= \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x[N_2 n_1 + n_2] \cdot \underbrace{W_N^{N_2 k_1 n_1}}_{W_{N_1}^{k_1 n_1}} \cdot W_N^{k_1 n_2} \cdot \underbrace{W_N^{k_1 N_1 n_2}}_{W_{N_2}^{k_2 n_2}} \cdot \underbrace{W_N^{N_1 N_2 k_2 n_1}}_{1} \\ &= \sum_{n_2=0}^{N_2-1} \left\{ \underbrace{\left[\underbrace{\sum_{n_1=0}^{N_1-1} x[N_2 n_1 + n_2] \cdot W_{N_1}^{k_1 n_1}}_{N_1\text{-point}} \right] \cdot \underbrace{W_N^{k_1 n_2}}_{\substack{\text{twiddle} \\ \text{factor}}} \right\} \cdot W_{N_2}^{k_2 n_2}}_{N_2\text{-point}} \end{aligned}$$

■ **Procedure**

(1) Compute N_1-point DFT: (row transform)

$$G[n_2, k_1] = \sum_{n_1=0}^{N_1-1} x[N_2 n_1 + n_2] \cdot W_{N_1}^{k_1 n_1}$$

(2) Multiply twiddle factors:

$$\tilde{G}[n_2, k_1] = W_N^{k_1 n_2} \cdot G[n_2, k_1]$$

(3) Compute N_2-point DFT: (column transform)

$$X[k_1 + N_1 k_2] = \sum_{n_2=0}^{N_2-1} \tilde{G}[n_2, k_1] \cdot W_{N_2}^{k_2 n_2}$$
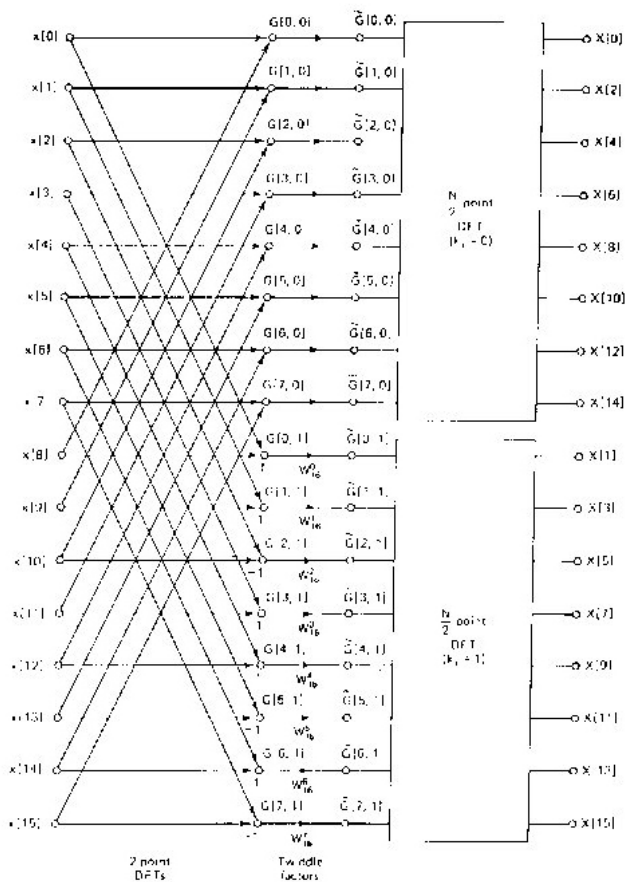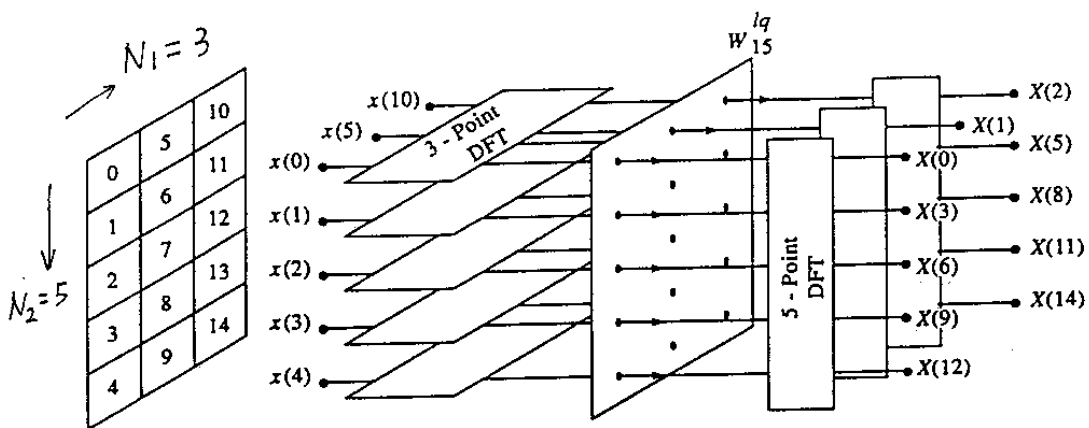
Figure 9.27 Flow graph of decomposition of an N-point DFT into two (N/2)-point DFTs (N = 16)

(Computation of *N*=15-point DFT by means of 3-point and 5-point DFTs.)

■ **Extension:** $N = N_1 N_2 \cdots N_\nu$

Let $\mu(N) \equiv$ number of multiplications for N - point DFT

If $N = N_1 N_2$

$$\begin{cases} 1.\,\text{row transform}: & N_2 \cdot \mu(N_1) \\ 2.\,\text{twiddle factors}: & N_1 N_2 = N \\ 3.\,\text{column transfrm}: & N_1 \cdot \mu(N_2) \end{cases}$$

$$\begin{aligned} \mu(N) &= N_2 \cdot \mu(N_1) + N_1 \cdot \mu(N_2) + N \\ &= N\left(\frac{\mu(N_1)}{N_1} + \frac{\mu(N_2)}{N_2} + 1\right) \end{aligned}$$

In general, $\mu(N) = N\left(\displaystyle\sum_{i=1}^{\nu} \frac{\mu(N_i)}{N_i} + (\nu - 1)\right)$

In fact, the term $(\nu - 1)$ should be $(\nu-1)\big/2$ because rearranging the butterfly structure

would make half of the branches becoming "1".

■ Special Case: $N_1 = N_2 = \cdots = N_\nu = 2$

Radix-2: $N_1 = N_2 = \cdots = N_\nu = 2$ and $\nu = \log_2 N$

$\mu(N) = N(\nu - 1)\big/2$ multiplications because $\mu(2)$ requires no multiplications.

Radix-4: $N_1 = N_2 = \cdots = N_\nu = 4$ and $\nu = \log_4 N$

$\mu(N) = N(\nu - 1)\big/2$ multiplications because $\mu(4)$ requires no multiplications. This FFT

has fewer stages than Radix-2 ==> fewer multiplications.

$$\mathbf{W}_4 = \begin{bmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^1 & W_4^2 & W_4^3 \\ W_4^0 & W_4^2 & W_4^4 & W_4^6 \\ W_4^0 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_4^1 & W_4^2 & W_4^3 \\ 1 & W_4^2 & W_4^0 & W_4^2 \\ 1 & W_4^3 & W_4^2 & W_4^1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

✧ Inverse FFT

- **IDFT:** $x[n] = \dfrac{1}{N} \displaystyle\sum_{k=0}^{N-1} X[k] \cdot W_N^{-kn}$ (*)

  DFT: $X[k] = \displaystyle\sum_{n=0}^{N-1} x[n] \cdot W_N^{nk}$

  Hence, take the conjugate of (*) :

  $$
  \begin{aligned}
  x^*[n] &= \frac{1}{N}\left(\sum_{k=0}^{N-1} X[k] \cdot W_N^{-kn}\right)^* \\
  &= \frac{1}{N}\sum_{k=0}^{N-1}\left(X[k] \cdot W_N^{-kn}\right)^* \\
  &= \frac{1}{N}\sum_{k=0}^{N-1}\left(X^*[k] \cdot W_N^{kn}\right) \\
  &= \frac{1}{N}\,\mathrm{DFT}\!\left[X^*(k)\right]
  \end{aligned}
  $$

  Take the conjugate of the above equation:

  $$
  \begin{aligned}
  x[n] &= \frac{1}{N}\left(\mathrm{DFT}\!\left[X^*(k)\right]\right)^* \\
  &= \frac{1}{N}\left(\mathrm{FFT}\!\left[X^*(k)\right]\right)^*
  \end{aligned}
  $$

  Thus, we can use the FFT algorithm to compute the inverse DFT.

✧ The Goertzel Algorithm
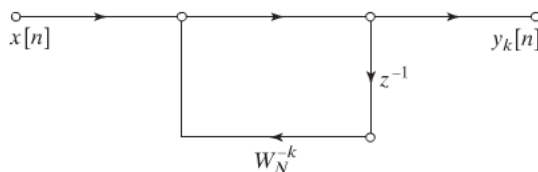
$$
W_N^{-kN} = e^{j(\frac{2\pi}{N})Nk} = e^{j2\pi k} = 1
$$

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

If we define x[n] = 0 for n < 0 and n ≥ N

and $y_k[n] = \displaystyle\sum_{r=-\infty}^{\infty} x[r]W_N^{-k(N-r)}u[n-r] = x[n] * (W_N^{-kn}u[n])$,

$$
\Rightarrow \quad X[k] = y_k[n]\big|_{n=N}
$$

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

If x[n] is complex, we need 4 real multiplications and 4 real additions to compute each $y_k[n]$.

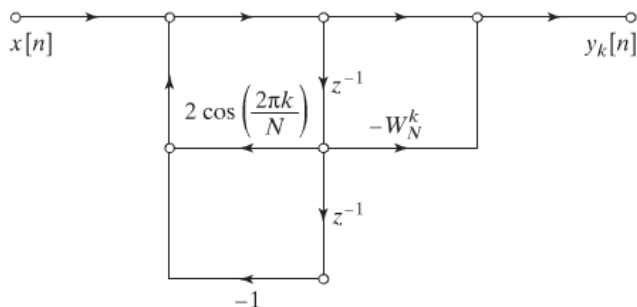To compute $y_k[N]$, we need to compute $y_k[1]$, $y_k[2]$, ..., $y_k[N-1]$.

$\Rightarrow$ We need 4N real multiplications and 4N real additions to compute X[k].

Remarks:

➢ less efficient than the direct method.

➢ Avoid the computation or storage of the coefficients W_N^{kn}.

To reduce the number of multiplications,

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

If x[n] is complex, we only need 2 real multiplications and 4 real additions to implement the poles of the system.

(The complex multiplication by $-W_N^k$ needs not be performed at every iteration.)

$\Rightarrow$ To compute X[k], we need 2N real multiplications and 4N real additions for the poles and 4 real multiplications and 4 real additions for the zero.

Remarks:

➢ Avoid the computation or storage of the coefficients W_N^{kn}.

➢ Only need to compute and save W_N^k and $\cos(2\pi k/N)$.