Chapter 9: Algorithmic Strength Reduction in Filters and Transforms

Keshab K. Parhi

Outline

- Introduction
- Parallel FIR Filters
  - Formulation of Parallel FIR Filter Using Polyphase Decomposition
  - Fast FIR Filter Algorithms
- Discrete Cosine Transform and Inverse DCT
  - Algorithm-Architecture Transformation
  - Decimation-in-Frequency Fast DCT for 2^M-point DCT

Introduction

- Strength reduction leads to a reduction in hardware complexity by exploiting substructure sharing and leads to less silicon area or power consumption in a VLSI ASIC implementation or less iteration period in a programmable DSP implementation
- Strength reduction enables design of parallel FIR filters with a less-than-linear increase in hardware
- DCT is widely used in video compression. Algorithm-architecture transformations and the decimation-in-frequency approach are used to design fast DCT architectures with significantly less number of multiplication operations

Parallel FIR Filters

Formulation of Parallel FIR Filters Using Polyphase Decomposition

- An N-tap FIR filter can be expressed in time-domain as

$$y(n) = h(n) * x(n) = \sum_{i=0}^{N-1} h(i)x(n-i), \quad n = 0,1,2,\cdots,\infty$$

  - where {x(n)} is an infinite length input sequence and the sequence $\{h(n)\}$ contains the FIR filter coefficients of length N
  - In Z-domain, it can be written as

$$Y(z) = H(z) \cdot X(z) = \left(\sum_{n=0}^{N-1} h(n)z^{-n} \right) \cdot \left(\sum_{n=0}^{\infty} x(n)z^{-n} \right)$$

- The Z-transform of the sequence x(n) can be expressed as:

$$\begin{aligned} X(z) &= x(0)+x(1)z^{-1}+x(2)z^{-2}+x(3)z^{-3}+\cdots \\ &= \left[x(0)+x(2)z^{-2}+x(4)z^{-4}+\cdots\right]+z^{-1}\left[x(1)+x(3)z^{-2}+x(5)z^{-4}+\cdots\right] \\ &= X_0(z^2)+z^{-1}X_1(z^2) \end{aligned}$$

  - where $X_0(z^2)$ and $X_1(z^2)$, the two polyphase components, are the z-transforms of the even time series $\{x(2k)\}$ and the odd time-series $\{x(2k+1)\}$, for $\{0 \le k < \infty\}$, respectively
- Similarly, the length-N filter coefficients H(z) can be decomposed as:

$$H(z) = H_0(z^2) + z^{-1}H_1(z^2)$$

  - where $H_0(z^2)$ and $H_1(z^2)$ are of length N/2 and are referred as even and odd sub-filters, respectively
- The even-numbered output sequence {y(2k)} and the odd-numbered output sequence {y(2k+1)} for $\{0 \le k < \infty\}$ can be computed as

(continued on the next page)

- (cont'd)

$$\begin{aligned} Y(z) &= Y_0(z^2) + z^{-1}Y_1(z^2) \\ &= \left(X_0(z^2) + z^{-1}X_1(z^2)\right)\cdot\left(H_0(z^2) + z^{-1}H_1(z^2)\right) \\ &= X_0(z^2)H_0(z^2) + z^{-1}\left[X_0(z^2)H_1(z^2) + X_1(z^2)H_0(z^2)\right] \\ &\quad + z^{-2}\left[X_1(z^2)H_1(z^2)\right] \end{aligned}$$

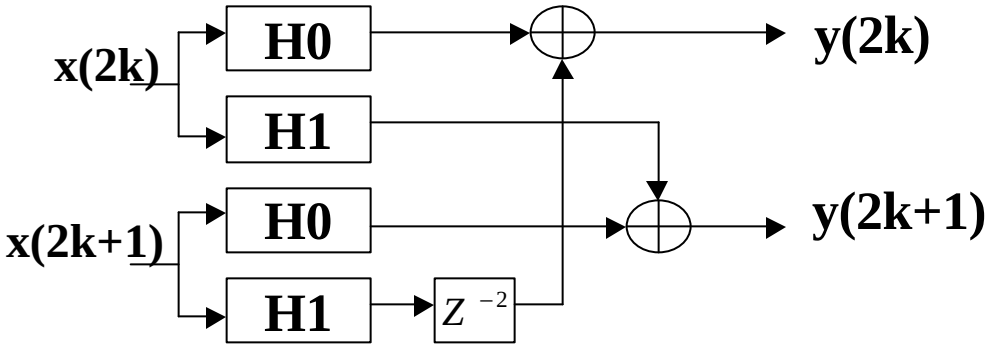
  - i.e.,

$$\begin{aligned} Y_0(z^2) &= X_0(z^2)H_0(z^2) + z^{-2}X_1(z^2)H_1(z^2) \\ Y_1(z^2) &= X_0(z^2)H_1(z^2) + X_1(z^2)H_0(z^2) \end{aligned}$$

  - where $Y_0(z^2)$ and $Y_1(z^2)$ correspond to y(2k) and y(2k+1) in time domain, respectively. This 2-parallel filter processes 2 inputs x(2k) and x(2k+1) and generates 2 outputs y(2k) and y(2k+1) every iteration. It can be written in matrix-form as:

$$Y = H\cdot X \quad \textbf{or} \quad \begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix} = \begin{bmatrix} H_0 & z^{-2}H_1 \\ H_1 & H_0 \end{bmatrix} \cdot \begin{bmatrix} X_0 \\ X_1 \end{bmatrix} \qquad (9.1)$$

– The following figure shows the traditional 2-parallel FIR filter structure, which requires 2N multiplications and 2(N-1) additions



- For 3-phase poly-phase decomposition, the input sequence X(z) and the filter coefficients H(z) can be decomposed as follows

$$X(z) = X_0(z^3) + z^{-1}X_1(z^3) + z^{-2}X_2(z^3),$$

$$H(z) = H_0(z^3) + z^{-1}H_1(z^3) + z^{-2}H_2(z^3)$$

– where $\{X_0(z^3), X_1(z^3), X_2(z^3)\}$ correspond to x(3k),x(3k+1) and x(3k+2) in time domain, respectively; and $\{H_0(z^3), H_1(z^3), H_2(z^3)\}$ are the three sub-filters of H(z) with length N/3.

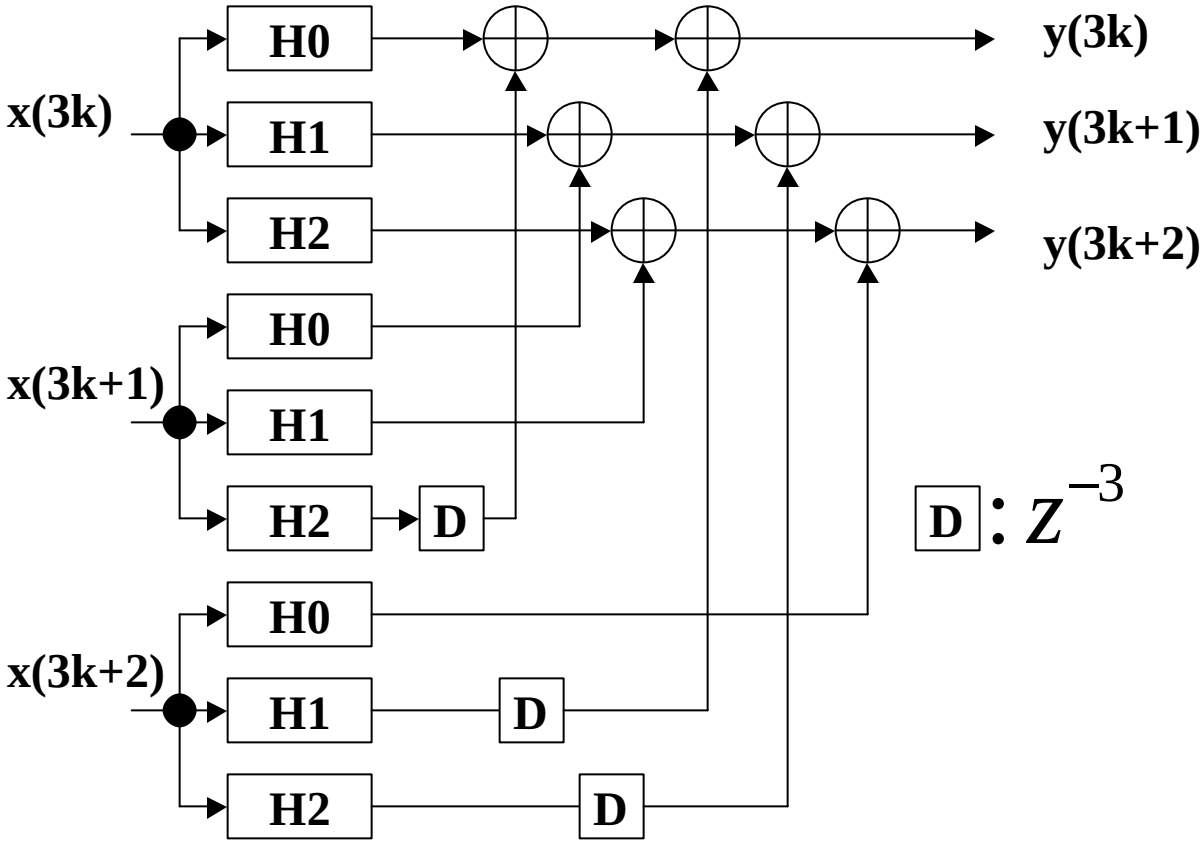
– The output can be computed as:

$$\begin{aligned} Y(z) &= Y_0(z^3) + z^{-1}Y_1(z^3) + z^{-2}Y_2(z^3) \\ &= \left(X_0 + z^{-1}X_1 + z^{-2}X_2\right)\cdot\left(H_0 + z^{-1}H_1 + z^{-2}H_2\right) \\ &= \left[X_0H_0 + z^{-3}\left(X_1H_2 + X_2H_1\right)\right] + z^{-1}\left[X_0H_1 + X_1H_0 + z^{-3}X_2H_2\right] \\ &\quad + z^{-2}\left[X_0H_2 + X_1H_1 + X_2H_0\right] \end{aligned}$$

– In every iteration, this 3-parallel FIR filter processes 3 input samples x(3k), x(3k+1) and x(3k+2), and generates 3 outputs y(3k), y(3k+1) and y(3k+2), and can be expressed in matrix form as:

$$\begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} H_0 & z^{-3}H_2 & z^{-3}H_1 \\ H_1 & H_0 & z^{-3}H_2 \\ H_2 & H_1 & H_0 \end{bmatrix} \cdot \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} \qquad (9.2)$$

– The following figure shows the traditional 3-parallel FIR filter structure, which requires 3N multiplications and 3(N-1) additions



- **Generalization:**
  - The outputs of an L-Parallel FIR filter can be computed as:

$$Y_k = z^{-L}\left(\sum_{i=k+1}^{L-1} H_i X_{L+k-i}\right) + \left(\sum_{i=0}^{k} H_i x_{k-i}\right), \quad 0 \le k \le L-2$$
$$Y_{L-1} = \sum_{i=0}^{L-1} H_i X_{L-1-i} \qquad (9.3)$$

  - This can also be expressed in Matrix form as

$$Y = H \cdot X$$
$$\begin{bmatrix} Y_0 \\ Y_1 \\ \cdots \\ Y_{L-1} \end{bmatrix} = \begin{bmatrix} H_0 & z^{-L}H_{L-1} & \cdots & z^{-L}H_1 \\ H_1 & H_0 & \cdots & z^{-L}H_2 \\ \cdots & \cdots & \cdots & \cdots \\ H_{L-1} & H_{L-2} & \cdots & H_0 \end{bmatrix} \cdot \begin{bmatrix} X_0 \\ X_1 \\ \cdots \\ X_{L-1} \end{bmatrix} \qquad (9.4)$$

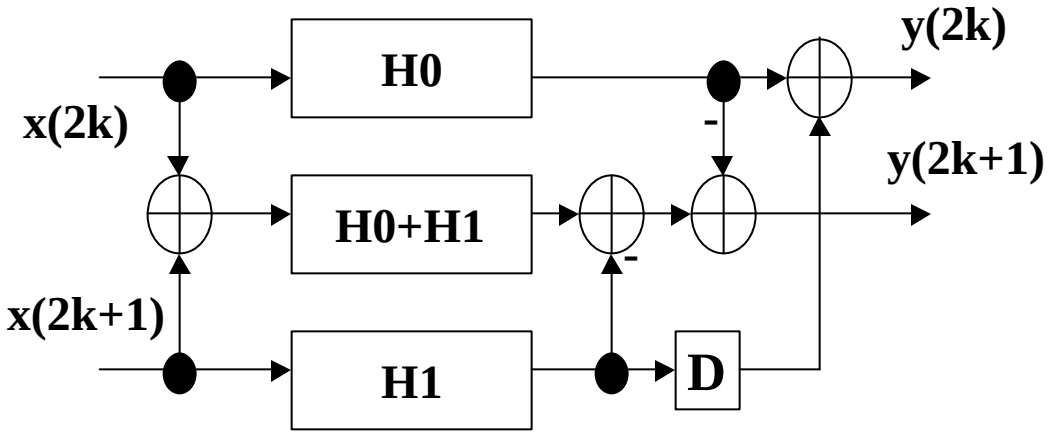
Note: H is a pseudo-circulant matrix

Two-parallel and Three-parallel Low-Complexity FIR Filters

- **Two-parallel Fast FIR Filter**
  - The 2-parallel FIR filter can be rewritten as

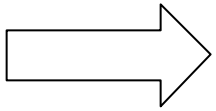
$$Y_0 = H_0 X_0 + z^{-2} H_1 X_1 \qquad (9.5)$$
$$Y_1 = (H_0 + H_1)\cdot(X_0 + X_1) - H_0 X_0 - H_1 X_1$$

  - This 2-parallel fast FIR filter contains 3 sub-filters. The 2 sub-filters H_0X_0 and H_1X_1 are shared for the computation of Y_0 and Y_1



- This 2-parallel filter requires 3 distinct sub-filters of length N/2 and 4 pre/post-processing additions. It requires 3N/2 = 1.5N multiplications and 3(N/2-1)+4=1.5N+1 additions. [The traditional 2-parallel filter requires 2N multiplications and 2(N-1) additions]
- Example-1: when N=8 and $H=\{h_0,h_1,\cdots,h_6,h_7\}$, the 3 sub-filters are

$$H_0=\{h_0,h_2,h_4,h_6\}$$



$$H_1=\{h_1,h_3,h_5,h_7\}$$

$$H_0+H_1=\{h_0+h_1,h_2+h_3,h_4+h_5,h_6+h_7\}$$

- The subfilter H_0+H_1 can be precomputed
- The 2-parallel filter can also be written in matrix form as

$$Y_2=Q_2\cdot H_2\cdot P_2\cdot X_2 \qquad (9.6)$$

Q_2 is a post-processing matrix which determines the manner in which the filter outputs are combined to correctly produce the parallel outputs and P_2 is a pre-processing matrix which determines the manner in which the inputs should be combined

– **(matrix form)**

$$\begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & z^{-2} \\ -1 & 1 & -1 \end{bmatrix} \cdot diag\left(\begin{matrix} H_0 \\ H_0 + H_1 \\ H_1 \end{matrix} \right) \cdot \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_0 \\ X_1 \end{bmatrix} \quad (9.7)$$

– where diag(h*) represents an NXN diagonal matrix H_2 with diagonal elements h*.

– **Note:** the application of FFA diagonalizes the original pseudo-circulant matrix H. The entries on the diagonal of H_2 are the sub-filters required in this parallel FIR filter

– Many different equivalent parallel FIR filter structures can be obtained. For example, this 2-parallel filter can be implemented using sub-filters $\{H_0, H_0 - H_1, H_1\}$ which may be more attractive in narrow-band low-pass filters since the sub-filter $H_0 - H_1$ requires fewer non-zero bits than $H_0 + H_1$. The parallel structure containing $H_0 + H_1$ is more attractive for narrow-band high-pass filters.

- **3-Parallel Fast FIR Filter**
  - A fast 3-parallel FIR algorithm can be derived by recursively applying a 2-parallel fast FIR algorithm and is given by

$$Y_0 = H_0X_0 - z^{-3}H_2X_2 + z^{-3}\left[(H_1+H_2)(X_1+X_2) - H_1X_1\right]$$

$$Y_1 = \left[(H_0+H_1)(X_0+X_1) - H_1X_1\right] - \left[H_0X_0 - z^{-3}H_2X_2\right] \qquad (9.8)$$

$$Y_2 = \left[(H_0+H_1+H_2)(X_0+X_1+X_2)\right] - \left[(H_0+H_1)(X_0+X_1) - H_1X_1\right] - \left[(H_1+H_2)(X_1+X_2) - H_1X_1\right]$$

  - The 3-parallel FIR filter is constructed using 6 sub-filters of length N/3, including H_0X_0, H_1X_1, H_2X_2, $(H_0+H_1)(X_0+X_1)$, $(H_1+H_2)(X_1+X_2)$ and $(H_0+H_1+H_2)(X_0+X_1+X_2)$
  - With 3 pre-processing and 7 post-processing additions, this filter requires 2N multiplications and 2N+4 additions which is 33% less than a traditional 3-parallel filter

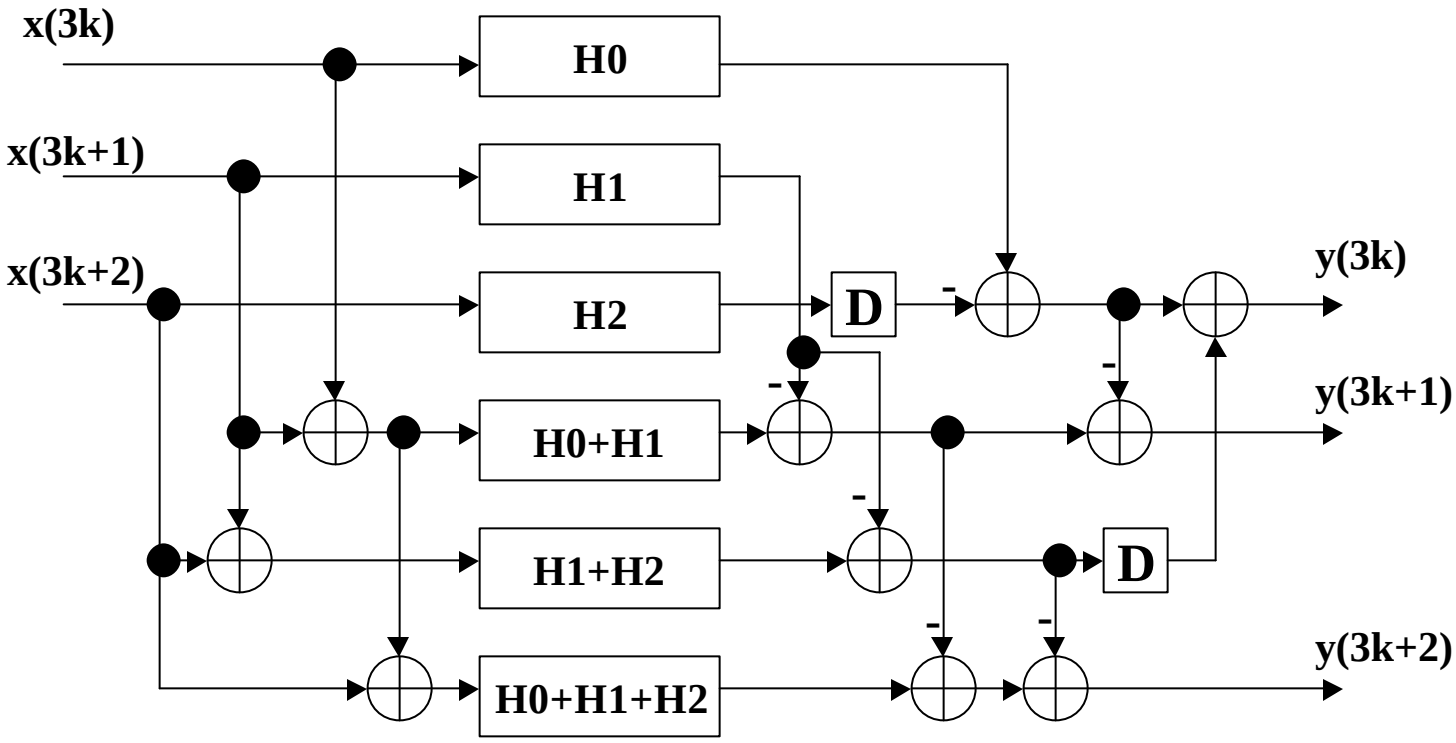
– The 3-parallel filter can be expressed in matrix form as

$$Y_3 = Q_3 \cdot H_3 \cdot P_3 \cdot X_3$$

$$Y_3 = \begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \end{bmatrix}, \quad Q_3 = \begin{bmatrix} 1 & 0 & z^{-3} & 0 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -z^{-3} & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H_3 = diag\begin{bmatrix} H_0 \\ H_1 \\ H_2 \\ H_0 + H_1 \\ H_1 + H_2 \\ H_0 + H_1 + H_2 \end{bmatrix}, \quad P_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad X_3 = \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} \qquad (9.9)$$

– Reduced-complexity 3-parallel FIR filter structure



Parallel FIR Filters (cont'd)

Parallel Filters by Transposition

- Any parallel FIR filter structure can be used to derive another parallel equivalent structure by transpose operation (or transposition). Generally, the transposed architecture has the same hardware complexity, but different finite word-length performance
- Consider the L-parallel filter in matrix form Y=HX (9.4), where H is an LXL matrix. An equivalent realization of this parallel filter can be generated by taking the transpose of the H matrix and flipping the vectors X and Y:

$$Y_F = H^T \cdot X_F$$

  - where $\begin{cases} X_F = [X_{L-1} \quad X_{L-2} \quad \cdots \quad X_0]^T \\ Y_F = [Y_{L-1} \quad Y_{L-2} \quad \cdots \quad Y_0]^T \end{cases}$ (9.10)

- Examples:
  - the 2-parallel FIR filter in (9.1) can be reformulated by using transposition as follows:

$$\begin{bmatrix} Y_1 \\ Y_0 \end{bmatrix} = \begin{bmatrix} H_0 & H_1 \\ z^{-2}H_1 & H_0 \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ X_0 \end{bmatrix}$$

  - Transposition of the 2-parallel fast filter in (9.6) leads to another equivalent structure:

$$Y_2 = Q_2 \cdot H_2 \cdot P_2 \cdot X_2 \Longrightarrow \begin{aligned} Y_{2_F} &= \left(Q_2 \cdot H_2 \cdot P_2\right)^T \cdot X_{2_F} \\ &= P_2^T \cdot H_2^T \cdot Q_2^T \cdot X_{2_F} \end{aligned}$$

$$\begin{bmatrix} Y_1 \\ Y_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \cdot diag\begin{pmatrix} H_0 \\ H_0 + H_1 \\ H_1 \end{pmatrix} \cdot \begin{bmatrix} 1 & -1 \\ 0 & 1 \\ z^{-2} & -1 \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ X_0 \end{bmatrix} \quad (9.11)$$

  - The reduced-complexity 2-parallel FIR filter structure by transposition is shown on next page

- **Signal-flow graph of the 2-parallel FIR filter**
- **Transposed signal-flow graph**

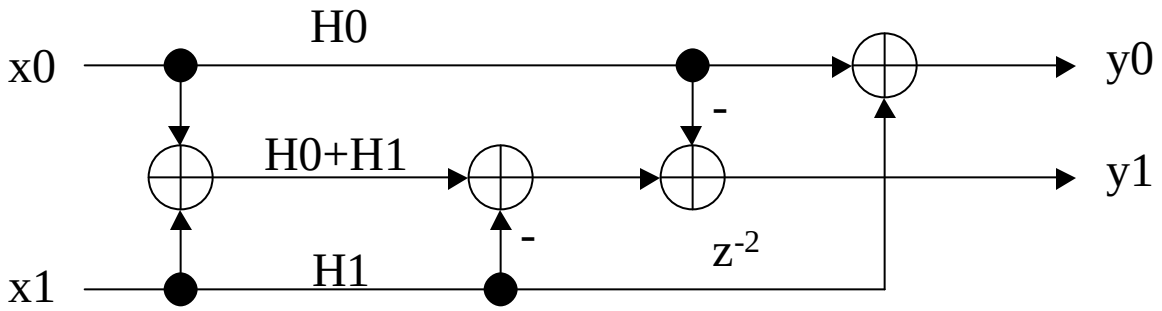


Fig. (a)

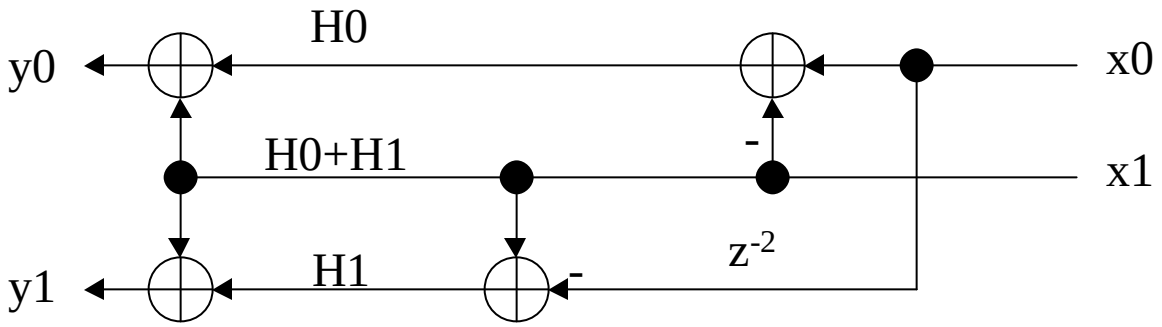


Fig. (b)

(c) Block diagram of the transposed reduced-complexity 2-parallel FIR filter

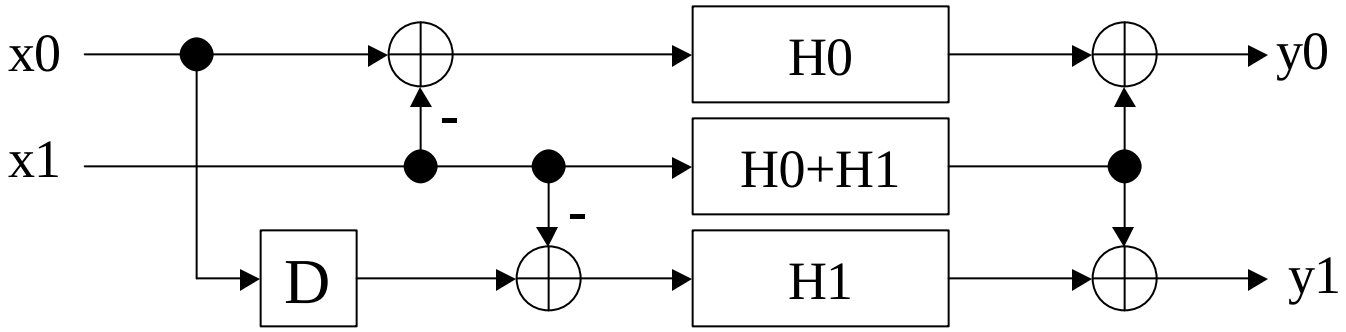


Fig. (c)

Parallel FIR Filters (cont'd)

Parallel Filter Algorithms from Linear Convolutions

- Any LXL convolution algorithm can be used to derive an L-parallel fast filter structure
- Example: the transpose of the matrix in a 2X2 linear convolution algorithm (9.12) can be used to obtain the 2-parallel filter (9.13):

$$\begin{bmatrix} s_2 \\ s_1 \\ s_0 \end{bmatrix} = \begin{bmatrix} h_1 & 0 \\ h_0 & h_1 \\ 0 & h_0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_0 \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix} = \begin{bmatrix} H_1 & H_0 & 0 \\ 0 & H_1 & H_0 \end{bmatrix} \cdot \begin{bmatrix} z^{-2} X_1 \\ X_0 \\ X_1 \end{bmatrix}$$

(9. 12) (9.13)

- Example: To generate a 2-parallel filter using 2X2 fast convolution, consider the following optimal 2X2 linear convolution:

$$s = C \cdot H \cdot A \cdot X$$

$$\begin{bmatrix} s_2 \\ s_1 \\ s_0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \cdot diag \begin{pmatrix} h_1 \\ h_0 + h_1 \\ h_0 \end{pmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_0 \end{bmatrix} \qquad (9.14)$$

  - **Note:** Flipping the samples in the sequences {s}, {h}, and {x} preserves the convolution formulation (i.e., the same **C** and **A** matrices can be used with the flipped sequences)
  - Taking the transpose of this algorithm, we can get the matrix form of the reduced-complexity 2-parallel filtering structure:

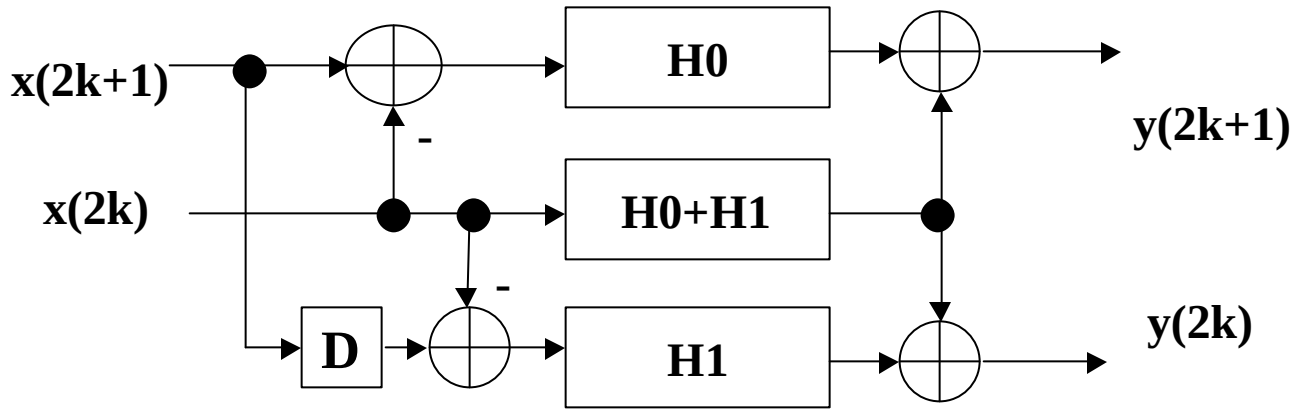
$$Y = \left(C \cdot H \cdot A\right)^T \cdot X = Q \cdot H \cdot P \cdot X$$

– The matrix form of the reduced-complexity 2-parallel filtering structure

$$\begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \cdot diag\left(\begin{matrix} H_1 \\ H_0 + H_1 \\ H_0 \end{matrix} \right) \cdot \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} z^{-2} X_1 \\ X_0 \\ X_1 \end{bmatrix} \quad (9.15)$$

– The 2-parallel architecture resulting from the matrix form is shown as follows

– **Conclusion:** this method leads to the same architecture that was obtained using the direct transposition of the 2-parallel FFA



Parallel FIR Filters (cont'd)

Fast Parallel FIR Algorithms for Large Block Sizes

- Parallel FIR filters with long block sizes can be designed by cascading smaller length fast parallel filters
- Example: an m-parallel FFA can be cascaded with an n-parallel FFA to produce an $(m \times n)$-parallel filtering structure. The set of FIR filters resulting from the application of the m-parallel FFA can be further decomposed, one at a time, by the application of the n-parallel FFA. The resulting set of filters will be of length $N/(m \times n)$.
- When cascading the FFAs, it is important to keep track of both the number of multiplications and the number of additions required for the filtering structure

– The number of required multiplications for an L-parallel filter with $L = L_1 L_2 \cdots L_r$ is given by:

$$M = \frac{N}{\prod_{i=1}^{r} L_i} \prod_{i=1}^{r} M_i \qquad (9.16)$$

- where r is the number of levels of FFAs used, L_i is the block size of the FFA at level-i, M_i is the number of filters that result from the applications of the i-th FFA and N is the length of the filter

– The number of required additions can be calculated as follows:

$$A = A_i \prod_{i=2}^{r} L_i + \sum_{i=2}^{r} \left[A_i \left(\prod_{j=i+1}^{r} L_j \right) \left(\prod_{k=1}^{i-1} M_k \right) \right] + \left(\prod_{i=1}^{r} M_i \right) \left(\frac{N}{\prod_{i=1}^{r} L_i} - 1 \right) \qquad (9.17)$$

- where A_i is the number of pre/post-processing adders required by the i-th FFA

– For example: consider the case of cascading two 2-parallel reduce-complexity FFAs, the resulting 4-parallel filtering structure would require a total of 9N/4 multiplications and 20+9(N/4-1) additions. Compared with the traditional 4-parallel filter which requires 4N multiplications. This results in a 44% hardware (area) savings

- **Example:** (Example 9.2.1, p.268) Calculating the hardware complexity
  – Calculate the number of multiplications and additions required to implement a 24-tap filter with block size of L=6 for both the cases $\{L_1 = 2, L_2 = 3\}$ and $\{L_1 = 3, L_2 = 2\}$:
    - For the case $\{L_1 = 2, L_2 = 3\}$:

$$M_1 = 3, \quad A_1 = 4, \quad M_2 = 6, \quad A_2 = 10,$$

$$M = \frac{24}{(2\times 3)} \times (3\times 6) = 72, \quad A = (4\times 3) + (10\times 3) + (3\times 6)\left[\frac{24}{(2\times 3)} - 1\right] = 96$$

- For the case $\{L_1 = 3, L_2 = 2\}$:

$$M_1 = 6, \quad A_1 = 10, \quad M_2 = 3, \quad A_2 = 4,$$

$$M = \frac{24}{(3\times 2)} \times (6\times 3) = 72, \quad A = (10\times 2) + (4\times 6) + (6\times 3)\left[\frac{24}{(3\times 2)} - 1\right] = 98$$

- How are the FFAs cascaded?
  - Consider the design of a parallel FIR filter with a block size of 4, using (9.3), we have

$$\begin{aligned} Y &= Y_0 + z^{-1}Y_1 + z^{-2}Y_2 + z^{-3}Y_3 \\ &= \left(X_0 + z^{-1}X_1 + z^{-2}X_2 + z^{-3}X_3\right)\cdot \\ &\quad \left(H_0 + z^{-1}H_1 + z^{-2}H_2 + z^{-3}H_3\right) \end{aligned} \qquad \textbf{(9.18)}$$

  - The reduced-complexity 4-parallel filtering structure is obtained by first applying the 2-parallel FFA to (9.18), then applying the FFA a second time to each of the filtering operations that result from the first application of the FFA
  - From (9.18), we have (see the next page):

– (cont'd) $Y=\left(X'_0+z^{-1}X'_1\right)\cdot\left(H'_0+z^{-1}H'_1\right)$

- where $\left\{\begin{array}{ll} X'_0=X_0+z^{-2}X_{2,} & X'_1=X_1+z^{-2}X_3 \\ H'_0=H_0+z^{-2}H_{2,} & H'_1=H_1+z^{-2}H_3 \end{array}\right\}$

– Application-1

$$Y=X_0^{'}H_0^{'}+z^{-1}\left[\left(X_0^{'}+X_1^{'}\right)\cdot\left(H_0^{'}+H_1^{'}\right)-X_0^{'}H_0^{'}-X_1^{'}H_1^{'}\right]+z^{-2}X_1^{'}H_1^{'} \quad (9.19)$$

- The 2-parallel FFA is then applied a second time to each of the filtering operations $\left\{X'_0H'_0, X'_1H'_1, \left(X'_0+X'_1\right)\cdot\left(H'_0+H'_1\right)\right\}$of (9.19)

– Application-2

- Filtering Operation $\left\{X'_0H'_0\right\}$

$$X_0^{'}H_0^{'}=\left(X_0+z^{-2}X_2\right)\left(H_0+z^{-2}H_2\right)$$

$$=X_0H_0+z^{-2}\left[\left(X_0+X_2\right)\cdot\left(H_0+H_2\right)-X_0H_0-X_2H_2\right]+z^{-4}X_2H_2$$

- Filtering Operation $\{X'_1 H'_1\}$

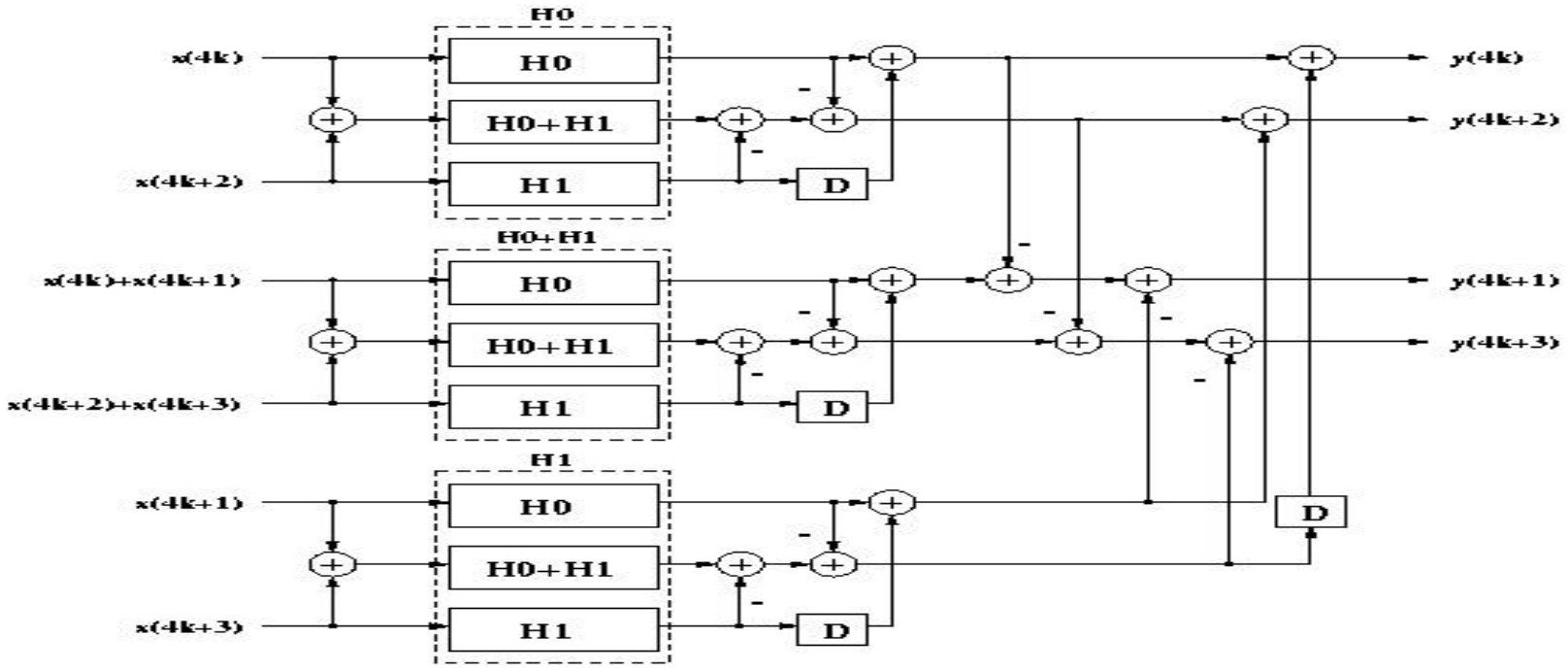
$$X'_1 H'_1 = (X_1 + z^{-2} X_3)(H_1 + z^{-2} H_3)$$
$$= X_1 H_1 + z^{-2}[(X_1 + X_3)\cdot(H_1 + H_3) - X_1 H_1 - X_3 H_3] + z^{-4} X_3 H_3$$

- Filtering Operation $\{(X'_0 + X'_1)(H'_0 + H'_1)\}$

$$(X'_0 + X'_1)(H'_0 + H'_1) = [(X_0 + X_1) + z^{-2}(X_2 + X_3)]\cdot[(H_0 + H_1) + z^{-2}(H_2 + H_3)]$$
$$= [(X_0 + X_1)(H_0 + H_1)] + z^{-4}[(X_2 + X_3)(H_2 + H_3)]$$
$$+ z^{-2}\begin{bmatrix}(X_0 + X_1 + X_2 + X_3)(H_0 + H_1 + H_2 + H_3)\\ -(X_0 + X_1)(H_0 + H_1) - (X_2 + X_3)(H_2 + H_3)\end{bmatrix}$$

– The second application of the 2-parallel FFA leads to the 4-parallel filtering structure (shown on the next page), which requires 9 filtering operations with length N/4

Reduced-complexity 4-parallel FIR filter (cascaded 2 by 2)



Discrete Cosine Transform and Inverse DCT

- The discrete cosine transform (DCT) is a frequency transform used in still or moving video compression. We discuss the fast implementations of DCT based on algorithm-architecture transformations and the decimation-in-frequency approach
- Denote the DCT of the data sequence x(n), n=0, 1,…, N-1, by X(k), k=0, 1, …, N-1. The DCT and inverse DCT (IDCT) are described by the following equations:
  - DCT:
    $$X(k) = e(k)\sum_{n=0}^{N-1} x(n)\cos\left[\frac{(2n+1)k}{2N}p\right], \quad k = 0,1,\cdots,N-1 \qquad (9.20)$$
  - IDCT:
    $$x(n) = \frac{2}{N}\sum_{k=0}^{N-1} e(k)X(k)\cos\left[\frac{(2n+1)k}{2N}p\right], \quad n = 0,1,\cdots,N-1 \qquad (9.21)$$

- where $$e(k) = \begin{cases} 1/\sqrt{2}, & k = 0 \\ 1, & otherwise \end{cases}$$

- Note: DCT is an orthogonal transform, i.e., the transformation matrix for IDCT is a scaled version of the transpose of that for the DCT and vice versa. Therefore, the DCT architecture can be obtained by "transposing" the IDCT, i.e., reversing the direction of the arrows in the flow graph of IDCT, and the IDCT can be obtained by "transposing" the DCT
- Direct implementation of DCT or IDCT requires N(N-1) multiplication operations, i.e., $O(N^2)$, which is hardware expensive.
- Strength reduction can reduce the multiplication complexity of a 8-point DCT from 56 to 13.

- Example (Example 9.3.1, p.277) Consider the 8-point DCT

$$X(k) = e(k)\sum_{n=0}^{7} x(n)\cos\left[\frac{(2n+1)k}{16}p\right], \quad k = 0,1,\cdots,7$$

$$\textit{where} \quad e(k) = \begin{cases} 1/\sqrt{2}, & k = 0 \\ 1, & \textit{otherwise} \end{cases}$$

  - It can be written in matrix form as follows: (where $c_i = \cos ip/16$)

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \\ X(5) \\ X(6) \\ X(7) \end{bmatrix} = \begin{bmatrix} c_4 & c_4 & c_4 & c_4 & c_4 & c_4 & c_4 & c_4 \\ c_1 & c_3 & c_5 & c_7 & c_9 & c_{11} & c_{13} & c_{15} \\ c_2 & c_6 & c_{10} & c_{14} & c_{18} & c_{22} & c_{26} & c_{30} \\ c_3 & c_9 & c_{15} & c_{21} & c_{27} & c_1 & c_7 & c_{13} \\ c_4 & c_{12} & c_{20} & c_{28} & c_4 & c_{12} & c_{20} & c_{28} \\ c_5 & c_{15} & c_{25} & c_3 & c_{13} & c_{23} & c_1 & c_{11} \\ c_6 & c_{18} & c_{30} & c_{10} & c_{22} & c_2 & c_{14} & c_{26} \\ c_7 & c_{21} & c_3 & c_{17} & c_{31} & c_{13} & c_{27} & c_9 \end{bmatrix} \cdot \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{bmatrix}$$

– The algorithm-architecture mapping for the 8-point DCT can be carried out in three steps
  - **First Step**: Using trigonometric properties, the 8-point DCT can be rewritten as in next page

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \\ X(5) \\ X(6) \\ X(7) \end{bmatrix} = \begin{bmatrix} c_4 & c_4 & c_4 & c_4 & c_4 & c_4 & c_4 & c_4 \\ c_1 & c_3 & c_5 & c_7 & -c_7 & -c_5 & -c_3 & -c_1 \\ c_2 & c_6 & -c_6 & -c_2 & -c_2 & -c_6 & c_6 & c_2 \\ c_3 & -c_7 & -c_1 & -c_5 & c_5 & c_1 & c_7 & -c_3 \\ c_4 & -c_4 & -c_4 & c_4 & c_4 & -c_4 & -c_4 & c_4 \\ c_5 & -c_1 & c_7 & c_3 & -c_3 & -c_7 & c_1 & -c_5 \\ c_6 & -c_2 & c_2 & -c_6 & -c_6 & c_2 & -c_2 & c_6 \\ c_7 & -c_5 & c_3 & -c_1 & c_1 & -c_3 & c_5 & -c_7 \end{bmatrix} \cdot \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{bmatrix} \quad (9.22)$$

– (continued)

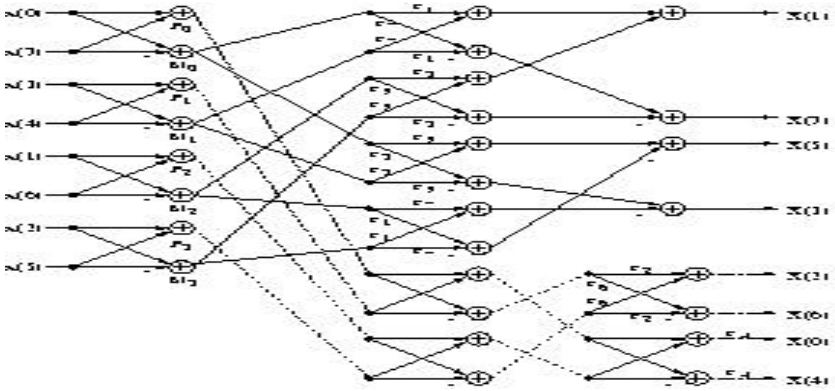
$$X(1) = M_0 c_1 + M_1 c_7 + M_2 c_3 + M_3 c_5, \quad X(2) = M_{10} c_2 + M_{11} c_6$$
$$X(7) = M_0 c_7 - M_1 c_1 + M_2 c_5 + M_3 c_3, \quad X(6) = M_{10} c_6 - M_{11} c_2$$
$$X(3) = M_0 c_3 - M_1 c_5 - M_2 c_7 - M_3 c_1, \quad X(4) = M_{100} \cdot c_4 \qquad (9.23)$$
$$X(5) = M_0 c_5 + M_1 c_3 - M_2 c_1 + M_3 c_7, \quad X(0) = P_{100} \cdot c_4$$

– where

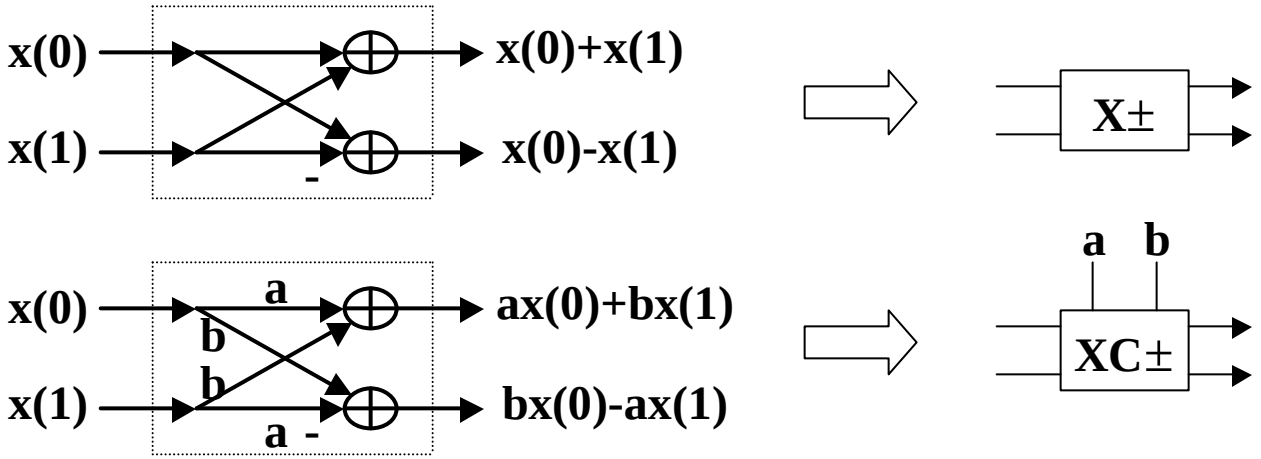
$$M_0 = x_0 - x_7, \quad M_1 = x_3 - x_4, \quad M_2 = x_1 - x_6, \quad M_3 = x_2 - x_5,$$
$$P_0 = x_0 + x_7, \quad P_1 = x_3 + x_4, \quad P_2 = x_1 + x_6, \quad P_3 = x_2 + x_5, \qquad (9.24)$$
$$M_{10} = P_0 - P_1, \quad M_{11} = P_2 - P_3, \quad P_{10} = P_0 + P_1, \quad P_{11} = P_2 + P_3,$$
$$M_{100} = P_{10} - P_{11}, \quad P_{100} = P_{10} + P_{11}$$

– The following figure (on the next page) shows the DCT architecture according to **(9.23) and (9.24)** with 22 multiplications.

Figure: The implementation of 8-point DCT structure in the first step (also see Fig. 9.10, p.279)

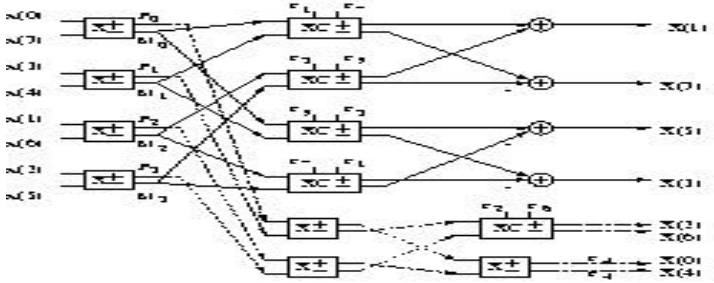


- **Second step**, the DCT structure (see Fig. 9.10, p.279) is grouped into different functional units represented by blocks and then the whole DCT structure is transformed into a block diagram
  - Two major blocks are defined as shown in the following figure

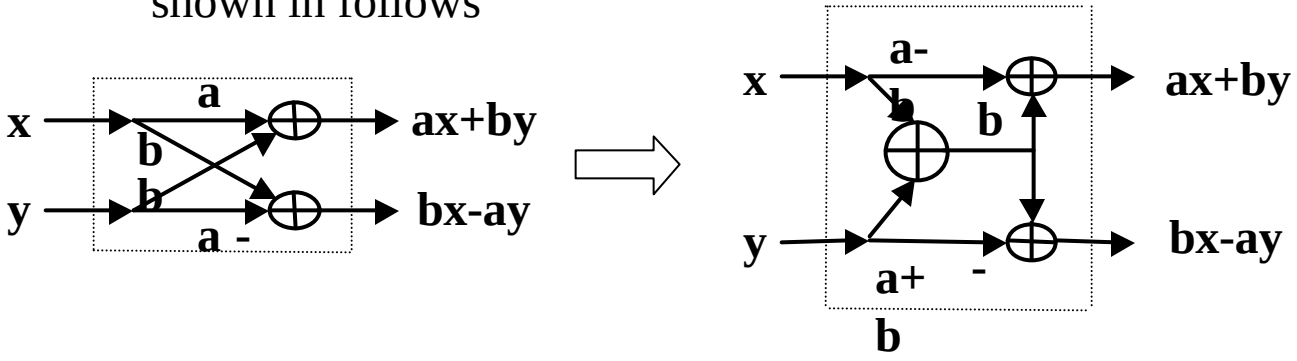


  - The transformed block diagram for an 8-point DCT is shown in the next page (also see Fig. 9.12 in p.280 of text book)

Figure: The implementation of 8-point DCT structure in the second step (also see Fig. 9.12, p.280)

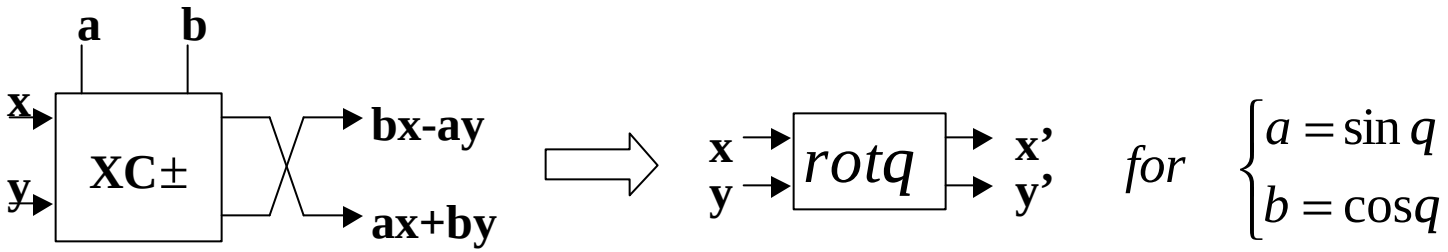


- **Third step**: Reduced-complexity implementations of various blocks are exploited (see Fig. 9.13, p.281)
  - The block **XC±** can be realized using 3 multiplications and 3 additions instead of using 4 multiplications and 2 additions, as shown in follows

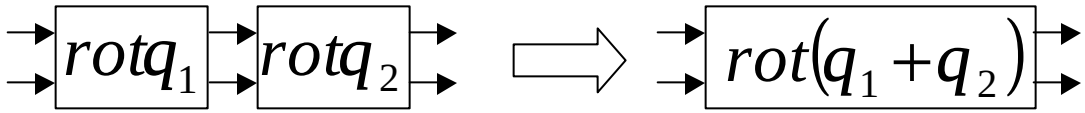


  - Define the block **XC±** with $\{a = \sin q, \quad b = \cos q\}$ and reversed outputs as a rotator block *rot q* that performs the following computation:

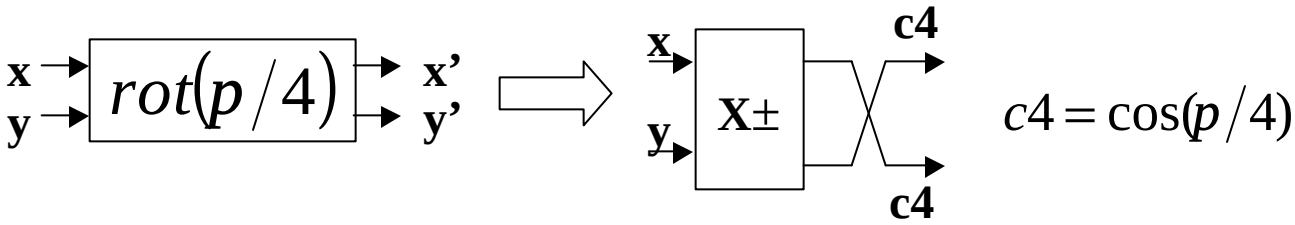
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos q & -\sin q \\ \sin q & \cos q \end{bmatrix} . \begin{bmatrix} x \\ y \end{bmatrix}$$



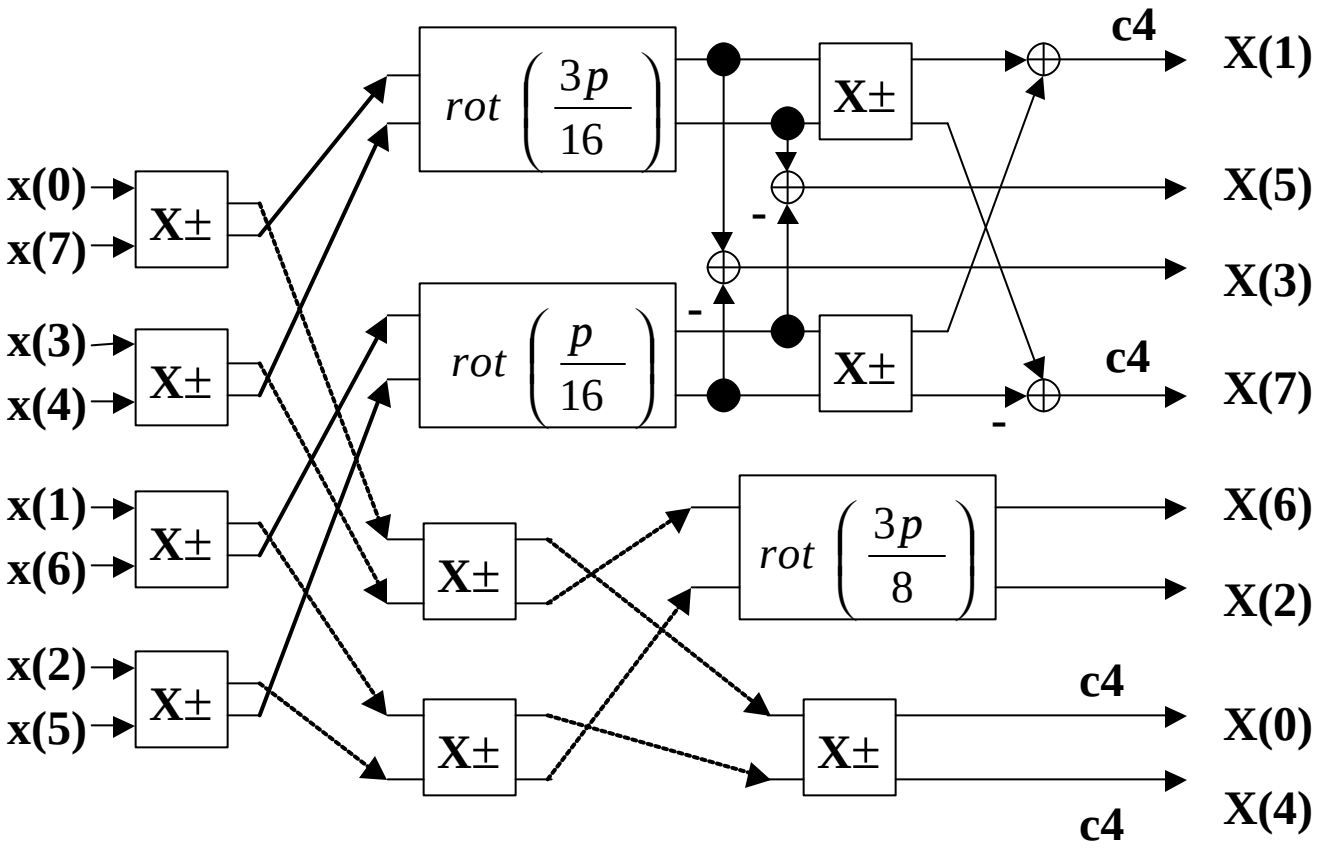
– Note: The angles of cascaded rotators can be simply added, as shown in the transformation block as follows:



– Note: Based on the fact that a rotator with $\{q = p/4\}$ is just like the block X± , we modify it as the following structure:



– From the three steps, we obtain the final structure where only 13 multiplications are required (also see Fig. 9.14, p.282)



Discrete Cosine Transform and Inverse DCT

Decimation-in-Frequency Fast DCT for 2^m-Point DCT

- The fast 2^m-point DCT/IDCT structures can be derived by the decimation-in-frequency approach, which is commonly used to derive the FFT structure to compute the discrete-Fourier transform (DFT). By power-of-2 decomposition, this algorithm reduces the number of multiplications to about

$$\{(N/2)\log_2 N\}$$

- We only derive the fast IDCT computation (The fast DCT structure can be obtained from IDCT by "transposition" according to their computation symmetry). For simplicity, the 2/N scaling factor in (9.21) is ignored in the derivation.

– Define $\hat{X}(k) = e(k) \cdot X(k)$ and decompose x(n) into even and odd indexes of k as follows

$$x(n) = \sum_{k=0}^{N-1} \hat{X}(k) \cos\left[\frac{(2n+1)k}{2N}\pi\right]$$

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

$$= \sum_{k=0}^{N/2-1} \hat{X}(2k) \cos\left[\frac{(2n+1)\pi(2k)}{2N}\right] + \frac{1}{2\cos[(2n+1)\pi/2N]} \bullet$$

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

– Notice

$$2\cos\frac{(2n+1)\pi(2k+1)}{2N} \cdot \cos\frac{(2n+1)\pi}{2N} = \cos\frac{(2n+1)\pi(k+1)}{N} + \cos\frac{(2n+1)\pi k}{N}$$

– Therefore, (since $\cos\left[(2n+1)p\left(N/2-1+1\right)/N\right]=0$)

$$\sum_{k=0}^{N/2-1} 2\hat{X}(2k+1)\cos\left[\frac{(2n+1)p(2k+1)}{2N}\right]\cos\left[\frac{(2n+1)p}{2N}\right]$$

$$=\sum_{k=0}^{N/2-1}\hat{X}(2k+1)\cos\left[\frac{(2n+1)p(k+1)}{N}\right]+\sum_{k=0}^{N/2-1}\hat{X}(2k+1)\cos\left[\frac{(2n+1)pk}{N}\right]$$

$$=\sum_{k=0}^{N/2-2}\hat{X}(2k+1)\cos\left[\frac{(2n+1)p(k+1)}{N}\right]+\sum_{k=0}^{N/2-1}\hat{X}(2k+1)\cos\left[\frac{(2n+1)pk}{N}\right]$$

– Substitute k'=k+1 into the first term, we obtain

$$\sum_{k=0}^{N/2-2}\hat{X}(2k+1)\cos\left[\frac{(2n+1)p(k+1)}{N}\right]$$

$$=\sum_{k'=1}^{N/2-1}\hat{X}(2k'-1)\cos\left[\frac{(2n+1)pk'}{N}\right]=\sum_{k=0}^{N/2-1}\hat{X}(2k'-1)\cos\left[\frac{(2n+1)pk'}{N}\right]$$

- where $\hat{X}(-1)=0$

– Then, the IDCT can be rewritten as

$$x(n) = \sum_{k=0}^{N/2-1} \hat{X}(2k)\cos\left[\frac{(2n+1)p k}{2(N/2)}\right] + \frac{1}{2\cos[(2n+1)p/2N]}\cdot$$

$$\sum_{k=0}^{N/2-1}\left[\hat{X}(2k+1)+\hat{X}(2k-1)\right]\cos\left[\frac{(2n+1)p k}{2(N/2)}\right]$$

– Define

$$\begin{cases} G(k) \equiv \hat{X}(2k), \\ H(k) \equiv \hat{X}(2k+1)+\hat{X}(2k-1), \end{cases} \quad k = 0,1,\cdots, N/2-1 \qquad (9.25)$$

– and

$$\begin{cases} g(n) \equiv \sum_{k=0}^{N/2-1} \hat{X}(2k)\cos\left[\frac{(2n+1)p k}{2(N/2)}\right], & n = 0,1,\cdots, N/2-1 \\ h(n) \equiv \sum_{k=0}^{N/2-1}\left[\hat{X}(2k+1)+\hat{X}(2k-1)\right]\cos\left[\frac{(2n+1)p k}{2(N/2)}\right] \end{cases} \qquad (9.26)$$

– Clearly, G(k) & H(k) are the DCTs of g(n) & h(n), respectively.

– Since $\begin{cases} \cos\dfrac{(2(N-1-n)+1)pk}{N} = \cos\dfrac{(2n+1)pk}{N} \\ \cos\dfrac{(2(N-1-n)+1)p}{N} = -\cos\dfrac{(2n+1)p}{N} \end{cases}$

– Finally, we can get

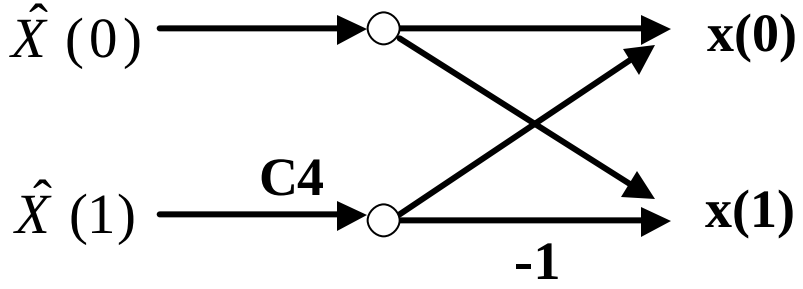
$$\begin{cases} x(n) = g(n) + \dfrac{1}{2\cos[(2n+1)p/(2N)]}h(n), \\ x(N-1-n) = g(n) - \dfrac{1}{2\cos[(2n+1)p/(2N)]}h(n), \end{cases} \quad n = 0,1,\cdots,N/2-1 \tag{9.27}$$

– Therefore, the N-point IDCT in (9.21) has been expressed in terms of two N/2-point IDCTs in (9.26). By repeating this process, the IDCT can be decomposed further until it can be expressed in terms of 2-point IDCTs. (The DCT algorithm can also be decomposed similarly. Alternatively, it can be obtained by transposing the IDCT)

- Example (see Example 9.3.2, p.284) Construct the 2-point IDCT butterfly architecture.
  - The 2-point IDCT can be computed as

$$\begin{cases} x(0) = \hat{X}(0) + \hat{X}(1)\cos(p/4), \\ x(1) = \hat{X}(0) - \hat{X}(1)\cos(p/4), \end{cases}$$

  - The 2-point IDCT can be computed using the following butterfly architecture



- Example (Example 9.3.3, p.284) Construct the 8-point fast DCT architecture using 2-point IDCT butterfly architecture.
  - With N=8, the 8-point fast DCT algorithm can be rewritten as:

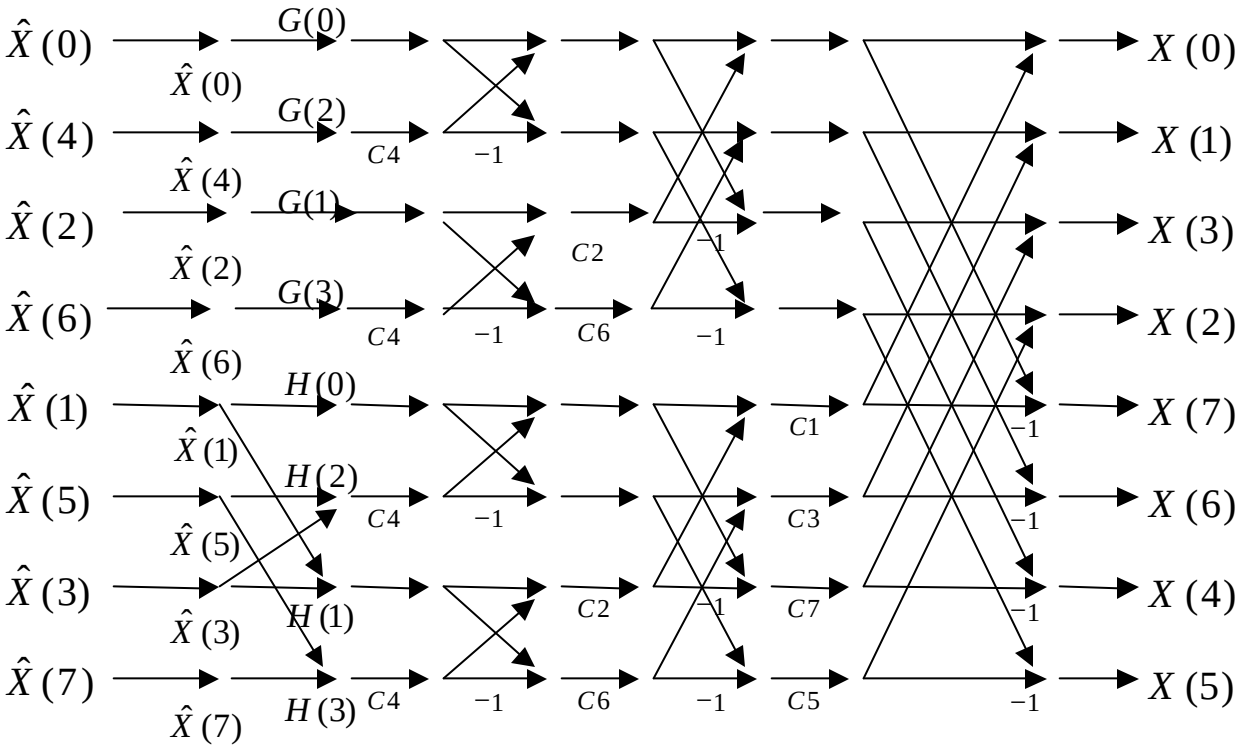
$$\begin{cases} G(k) \equiv \hat{X}(2k), \\ H(k) \equiv \hat{X}(2k+1) + \hat{X}(2k-1), \end{cases} \qquad k = 0,1,2,3$$

  - and

$$\begin{cases} g(n) = \sum_{k=0}^{3} G(k) \cos\left[\frac{(2n+1)\pi k}{8}\right], \quad n = 0,1,\cdots, N/2-1 \\ h(n) = \sum_{k=0}^{3-1} H(k) \cos\left[\frac{(2n+1)\pi k}{8}\right] \end{cases}$$

$$\begin{cases} x(n) = g(n) + \frac{1}{2\cos[(2n+1)\pi/16]} h(n), \\ x(N-1-n) = g(n) - \frac{1}{2\cos[(2n+1)\pi/16]} h(n), \end{cases}$$

– The 8-point fast IDCT is shown below (also see Fig.9.16, p.285), where only 13 multiplications are needed. This structure can be transposed to get the fast 8-point DCT architecture as shown on the next page (also see Fig. 9.17, p.286) (Note: for N=8, $C4 = 1/[2\cos(4p/16)] = \cos(p/4)$ in both figures)



Fast 8-point DCT Architecture