S-38.220
Postgraduate Course on Signal Processing in Communications, FALL - 99

Algorithmic strength reduction in filters and Transforms

Yaohui Liu

Laboartory of telecommunications Technology
Helsinki University of Technology
P.O.Box 3000 02150 Espoo

Email: yaohui.liu@hut.fi

Date: 08.11.1999

ABSTRACT

This paper exploits the interplay between the design of DSP algorithm and integrated circuit implementations. The design of algorithm structures for various DSP algorithms are discussed based on the algorithmic strength reduction transformations. Particularly, strength reduction techniques are applied to parallel fast filter, discrete cosine transforms (DCTs) and parallel rank-order filters.

1. INTRODUCTION

Digital signal processing (DSP) is widely used in applications such as video coding, noise cancellation and so on. The field of DSP has always been driven by the advances in DSP applications and in scaled very-large-scale-integrated (VLSI) technologies. Therefore, at any given time, DSP applications impose several challenges on the implementations of the DSP systems. These implementations must satisfy the enforced sampling rate constraints of the real-time DSP applications and must require less space and power consumption.

The report surveys the ideas of strength reduction in high-level algorithm transformations, which can be applied to reduce the number of multiplication in parallel FIR filter architecture, discrete cosine transforms (DCTs) and parallel rank-order filters.

This report is based on the book Chapter 9 of the book [1]. The multirate signal processing concepts and notations appeared in this report can refer to the chapter 8 of book [2]. The algorithms used in strength reduction include *Polyphase decomposition*, and a couple of Fast FIR algorithms for parallel FIR filtering; *Algorithm Architecture Transformation* and *Decimation-in-frequency* for DCT transform; *Odd-Even Merge-Sort algorithm* and *Running order merge sorter* for rank order filters. The design of fast Fourier transform (FFT) structures is also based on strength reduction transformations but is not covered in this report since it is covered in many introductory DSP textbooks, e.g. [2].

2. PARALLEL FIR FILTER

FIR filter is one of the most popular elements in DSP systems. In some application, FIR filters need to work at high frequencies, while in other applications, FIR filter circuit has to be low-power consuming, and operate at moderate frequencies. Parallel processing can be applied to achieve the above goal. However, the direct implementation of FIR filter will cause the area of the circuit increases linearly with the block size of the original circuit. Therefore, it is advantageous to realize parallel FIR filtering structure that consumes less area than the traditional parallel filters while having the same performance.

2.1 Formulating Parallel FIR filtering using Polyphase decomposition

In this section, we first review a method to realize the parallel structure of the FIR filters. In fact, it is a technique used in multirate signal processing.

The main idea of polyphase decomposition is that using the property of the transformation between the signal in time domain and in z-domain. A input sequence can be decomposed into even numbered part and odd-number part. The z-domain expression of the convolution of the signal and the FIR filter can be expressed as

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

where $X_0(z^2)$ and $X_1(z^2)$ are the z-transform of $x(2k)$ and $x(2k+1)$, respectively. Similarly, the N tap FIR filter coefficients $H(Z)$ can be decomposed as

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

The filter can process two inputs and generate two outputs per iteration. So that we have 2-parallel FIR filter, which can be written as

$$\begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix} = \begin{bmatrix} H_0 & z^{-2}H_1 \\ H_1 & H_0 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \end{bmatrix} \tag{3}$$

This polyphase decomposing process is shown in the following figure.

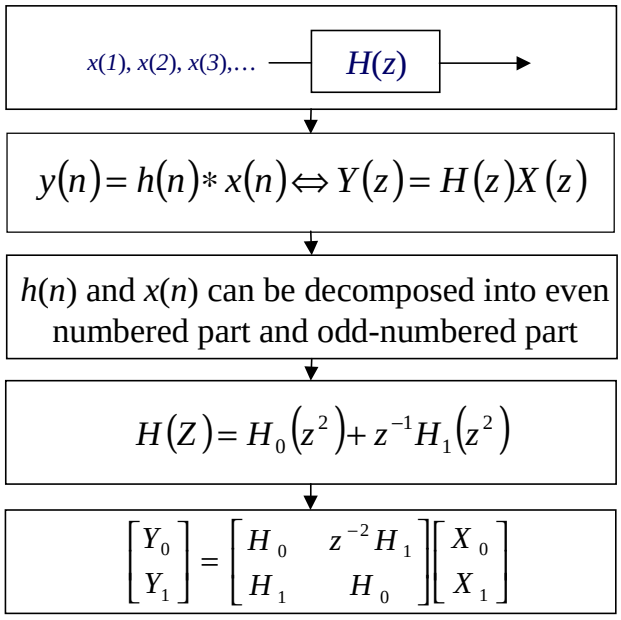

Figure 2-1: polyphase decomposition.

The implementation of the 2-parallel FIR filter is shown below,

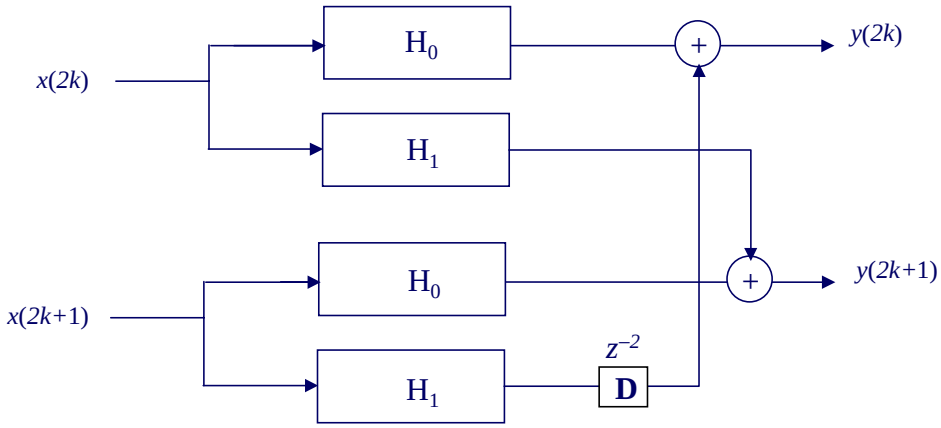

Figure 2-2: Traditional 2-parallel FIR filter implementation

Generally, the polyphase decomposition can be used to derive *L*-parallel FIR filters by decomposing $X(z)$, $H(z)$ and $Y(z)$ into L subsequences. As we see, such *L*-parallel filter doesn't save operations. It requires N/L multiply-add operations, which is linear in the block size *L*. We need to derive fast FIR algorithms to reduce the complexity for parallel FIR filters.

2.2 Fast FIR algorithm

According to Winograd [3], L-parallel FIR filters can be realized using approximately (2L-1) FIR filters of length-N/L In this section, we address the derivation of hardware efficient parallel FIR filters using the fast FIR algorithms.

2.2.1 Two parallel low complexity FIR filters

The 2-parallel FIR filter shown in figure 2-2 requires 2N multiplication and addition operation. If we rewrite equation (3) into

$$\begin{aligned} Y_0 &= H_0 X_0 + z^{-2} H_1 X_1 \\ Y_1 &= (H_0 + H_1) - H_0 X_0 - H_1 X_1 \end{aligned} \tag{4}$$

We get the reduced-complexity 2-parallel FIR filter.

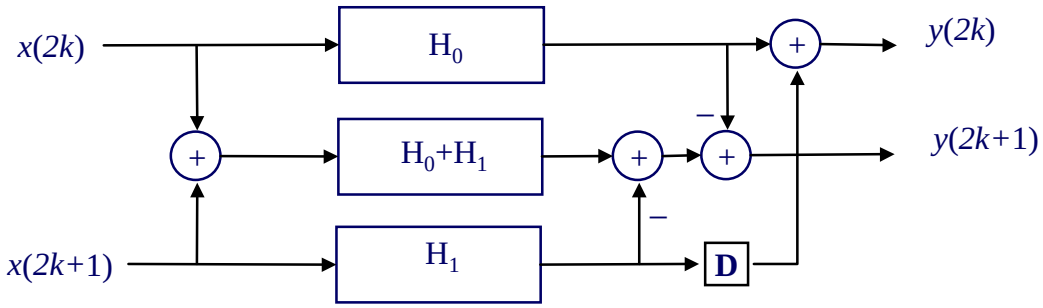

Figure 2-3: reduced-complexity 2-parallel FIR filter implementation

The reduction of the operation is summarized in the following table.

Table 2-1 The reduction of the operations

	Polyphase decomposition	***Low-complexity 2-parallel filter***
Filter-taps	$H_0=\{h_0, h_2, h_4, h_6\}$; $H_1=\{h_1, h_3, h_5, h_7\}$	$H_0 + H_1 = \{h_0+h_1, h_2+h_3, h_4+h_5, h_6+h_7\}$; $H_0=\{h_0, h_2, h_4, h_6\}$; $H_1=\{h_1, h_3, h_5, h_7\}$
×	$4N/2 = 2N$	$3N/2 = 1.5N$
+	$4(N/2-1) +2 = 2(N-1)$	$3(N/2-1) + 4 = 1.5N +1$

The reduction of operation for 2-parallel filters is that it requires 12 multiplications and 13 additions. We note that H_0+H_1 are precomputated

The 2 parallel filter can be expressed in matrix format. We can summarize that the reduction of operation is achieved by diagonalize the pseudocirculant matrix. As we see the **H** matrix has been diagonized in the following equation, so that the operation is reduced.

$$\begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & z^{-2} \\ -1 & 1 & -1 \end{bmatrix} diag \begin{bmatrix} H_0 \\ H_0 + H_1 \\ H_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \end{bmatrix} \quad (5)$$

i.e.

$$\mathbf{Y_2 = Q_2 H_2 P_2 X_2} \quad (6)$$

In the equation, Q2 and P2 are the postprocessing and preprocessing matrixs respectively.

We need to point out that there is not only one implementation of diagonized **H** matrix, if we precompute $H_0–H_1$, it is also an alternative to reduce the complexity of the operation. Certainly, we have another advantage of computing $H_0–H_1$, which has better wordlength effects.

2.2.2 Parallel filters by transposition

As point out in the last section, we can transpose the matrix to find another implementation of the Fast FIR algorithms. Although the transposed version of the implementation have the same hardware complexity, they have different wordlength and round off noise performance.
We can write the L-parallel FIR filter in matrix format

$$\mathbf{Y = HX} \quad (7)$$

We can perform the transposition into two steps

Setp 1: Transpose **H** matrix and flipping **X**, **Y**.
Setp 2: Transpose the signal flow graph (SFG)

As an example,

$$\mathbf{Y_2 = Q_2 H_2 P_2 X_2} \quad \Longrightarrow \quad \mathbf{Y_{2F} = \left(Q_2 H_2 P_2\right)^T X_{2F} = P_2^T H_2^T Q_2^T X_{2F}} \quad (8)$$

The SFG is shown below,

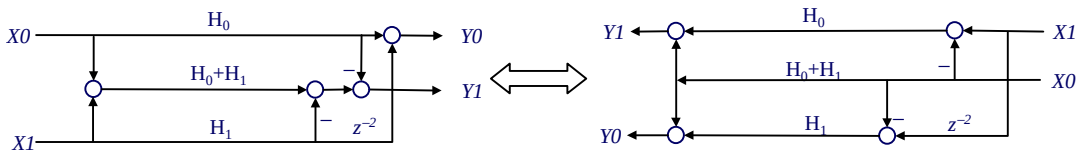

Figure 2-4 parallel filters by transposition

Generally, both matrix transposition and SFG transposition are applicable to any FFA to generate equivalent parallel filtering structures.

2.2.3 Parallel filters from linear convolution

Any L×L convolution algorithm can also be used to derive an L parallel fast filter structure. The reason is that the similarity of the transpose form of a linear convolution to the standard parallel filtering algorithm. The similarity is shown in the following equations.

$$\begin{bmatrix} s_2 \\ s_1 \\ s_0 \end{bmatrix} = \begin{bmatrix} h_1 & 0 \\ h_0 & h_1 \\ 0 & h_0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_0 \end{bmatrix} \tag{9}$$

$$\begin{bmatrix} Y_1 \\ Y_0 \end{bmatrix} = \begin{bmatrix} H_1 & H_0 & 0 \\ 0 & H_1 & H_0 \end{bmatrix} \begin{bmatrix} z^{-2}X_1 \\ X_0 \\ X_1 \end{bmatrix} \tag{10}$$

The basic idea of using such similarity to generate FFA is to start from an optimal linear convolution and take its transposition. As an example, the following optimal 2X2 optimal linear convolution:

$$\begin{bmatrix} s_2 \\ s_1 \\ s_0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} diag \begin{bmatrix} h_0 \\ h_0 + h_1 \\ h_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_0 \end{bmatrix} \tag{11}$$

We can flip the samples in the sequence of {s}, {h} and {x}, preserve the convolution formulation. Taking the transpose of this algorithm and by proper substitution for the elements in the sequence, We have

$$\begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} diag \begin{bmatrix} H_1 \\ H_0 + H_1 \\ H_0 \end{bmatrix} \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} z^{-1}X_1 \\ X_0 \\ X_1 \end{bmatrix} \tag{12}$$

The coresponding SFG has been shown in Figure 2-3.

2.2.4 Fast Parallel filters for large block sizes

The basic idea of this chapter is show how to implement an m-parallel FFA by cascading n-parallel FFA to produce an (m×n)-parallel filtering structure.

When we consider equation (7) again. For example, we design a 4-parallel FIR filter. We have

$$\begin{aligned} Y &= Y_0 + z^{-1}Y_1 + z^{-2}Y_2 + z^{-3}Y_3 \\ &= \left(X_0 + z^{-1}X_1 + z^{-2}X_2 + z^{-3}X_3\right)\left(H_0 + z^{-1}H_1 + z^{-2}H_2 + z^{-3}H_3\right) \end{aligned} \tag{13}$$

We can first apply the 2-parallel FFA to Equation (13), and then applying the FFA a second time to each of the filtering operations that result from the first application of the FFA. The processe is shown in the following figure

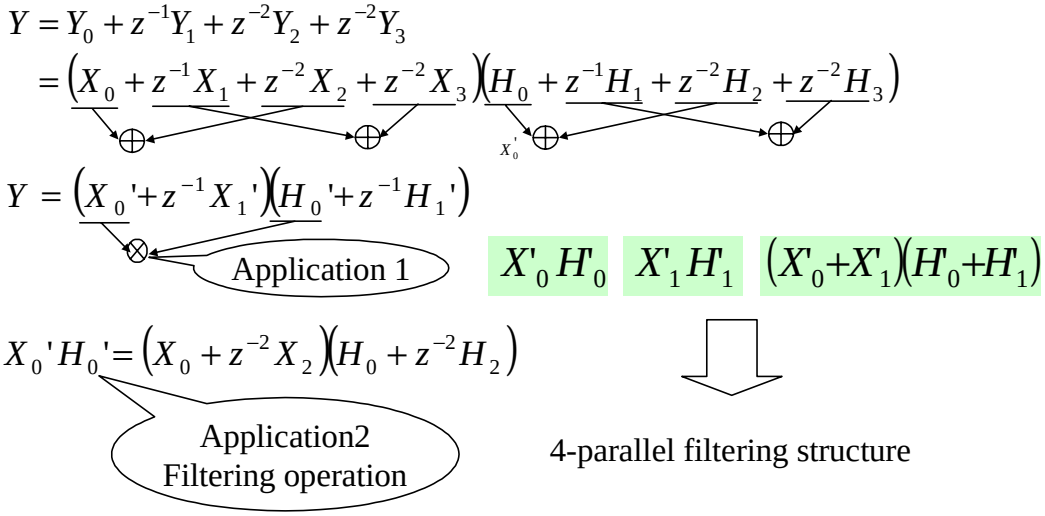

Figure 2-5: cascading structure for large block of FFA

By applying the operation shown above, we could have reduced complexity 4-parallel FIR filter shown below.

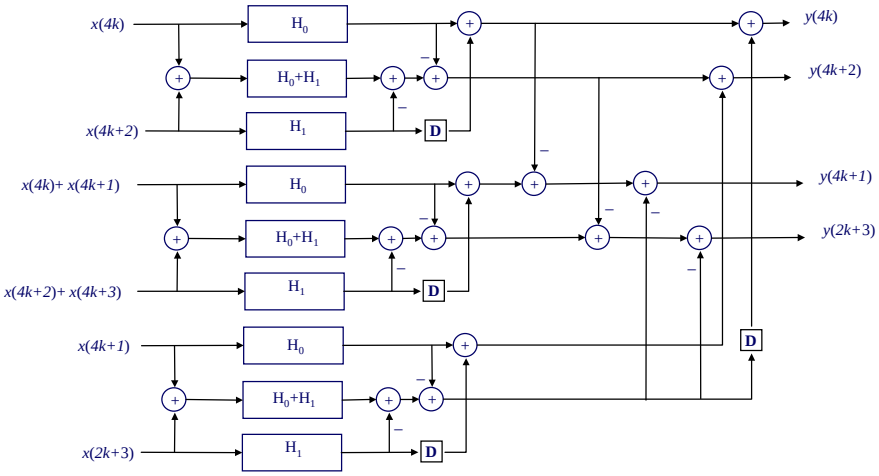

Figure 2-6: reduced complexity 4-parallel FIR filter

3. DISCRETE COSINE TRANSFORM AND INVERSE DCT

DCT is a very useful transformation in Video processing. We discuss how can we efficiently implement DCT transform. First, we have a look at the property of the DCT transform. The DCT and IDCT algorithms can be expressed by the following equations

DCT:

$$X(k) = e(k)\sum_{n=0}^{N-1} x(n)\cos\left[\frac{(2n+1)\pi k}{2N}\right], \quad k = 0,1,\ldots,N-1 \tag{14}$$

And IDCT

$$x(n) = \frac{2}{N}\sum_{k=0}^{N-1} e(k)X(k)\cos\left[\frac{(2n+1)\pi k}{2N}\right], \quad n = 0,1,\ldots,N-1 \tag{15}$$

Where

$$e(k) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if k = 0} \\ 1 & \text{otherwise} \end{cases} \tag{16}$$

DCT is an orthogonal transform. Reversing the firection of the arrows in the flow graph of IDCT, we can get DCT. Direct implementation of the DCT require $N(N-1)$ multiplication operations, i.e., $O(N^2)$.

3.1 Algorithm-architecture transformation

We can use algorithm-architecture transformation to reduce the number of multiplication, so that we can implement DCT efficiently. The reduandancy of the operation can be easily seen in the matrix expression of the algorithm. When we take 8 point DCT as an example

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

where

$$c_i = \cos\frac{i\pi}{16} \tag{18}$$

We can take 3 steps to perform the algorithm architecture mapping.

Step 1: Using the trigonometric property of the matrix, modify the DCT algorithm systemically to make it simpler.

Step 2: Group the DCT structure into different functional blocks. See as figure 3-1 shows.

Step 3: Reduce the complexity of each block.

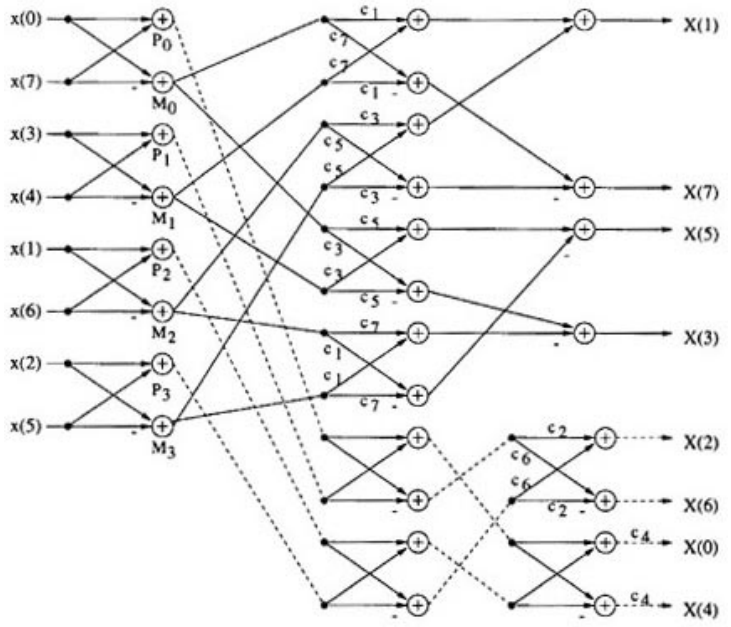

Figure 3-1: First step 8-point DCT structures

After these steps, the number of multiplication operations for 8-point DCT is reduced from 56 to 13. As shown in Figure 3-2.

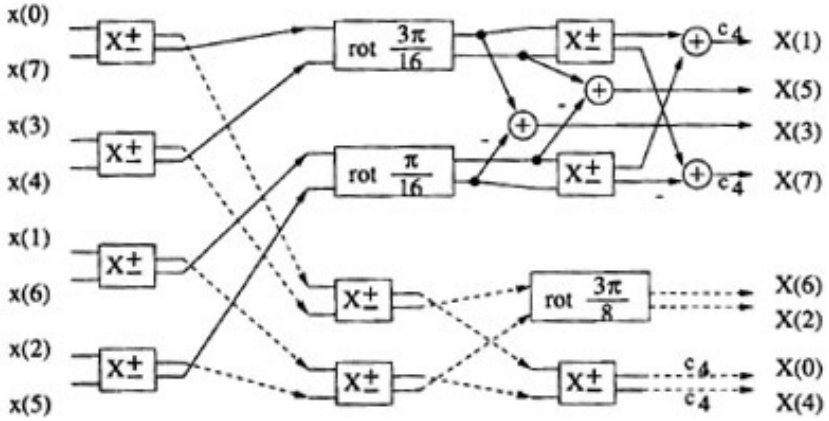

Figure 3-2: Final 8-point DCT structures

3.2 Decimation-in-frequency

Fast implementation of DCT/IDCT for 2^m-point can be derived by the decimation in frequency approach. In fact, the idea is similar to the Fast algorithm for DFT. FFT has been widely used in the current DSP implementations. If we consider a simplified IDCT as an example, we can ignore the scaler 2/N, Let

$$\hat{X}(k) = e(k)X(k) \tag{19}$$

We can write IDCT as

$$x(n) = \sum_{k=0}^{N-1} \hat{X}(k)\cos\left[\frac{(2n+1)\pi k}{2N}\right], \quad n = 0,1,...,N-1 \tag{20}$$

Define

$$\begin{aligned} G(k) &= \hat{X}(k) \\ H(k) &= \hat{X}(2k-1) + \hat{X}(2k-1) \\ for \;\; k &= 0,1,..., N/2-1 \end{aligned} \tag{21}$$

We can decompose N-point IDCT into two N/2 point IDCT, continue the process, we can finally get 2-point IDCT show in the following figure.

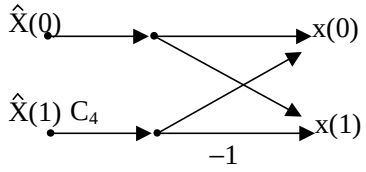

Figure 3-3: Two point IDCT butterfly architecture

Where $C_4=\cos(\pi/4)$.
13 multiplications are needed for fast 8 point IDCT. The flowgraph is shown in the following figure. The structure can be transposed to fet the fast 8-point DCT architecture.

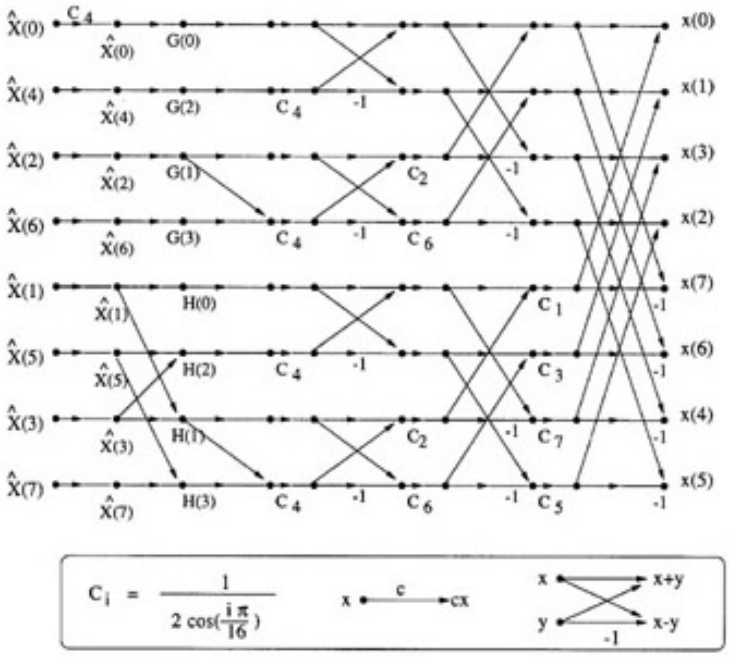

Figure 3-4: 8 point IDCT butterfly architecture

4. PARALLEL ARCHITECTURES FOR RANK-ORDER FILTERS

Rank-order filter (ROF) is a kind of nonlinear filter useful in canceling non-Gaussian noise. ROF sorts the input sequence and choose an output based on its rank. The system diagram is shown in Figure 4-1 with window size *W*=5.

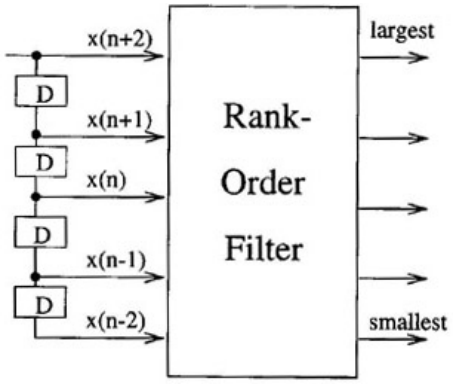

Figure 4-1: a rank-order filter with window size W=5 [1]

We discuss the parallel architecture of ROF design based on the Batcher's odd-Even Merge-Sort Algorithm and algorithmic strength reduction. By sharing some of the merge units in different blocks, the hardware complexity can be reduced.

4.1 Batcher's odd-Even Merge-Sort Algorithm

The odd-Even Merge-Sort Algorithm processes 2 presorted sequence and merge them into 1 sorted sequence. Figure 4-1 show a 4×4 merge-sort circuit. All inputs with odd subscripts are input to the odd merge. All inputs with even subscripts are input to the even merge. The output of each merge unit are again compared with each other except the first output of the odd-merge, which is the largest among all the inputs and the last output of the even merge, which is the smallest. As the figure shows, Large merge unit can be built using small merge-sort units. The smallest merge-sort unit is a 1×1 merge sort unit, which is usually denoted as C&S.

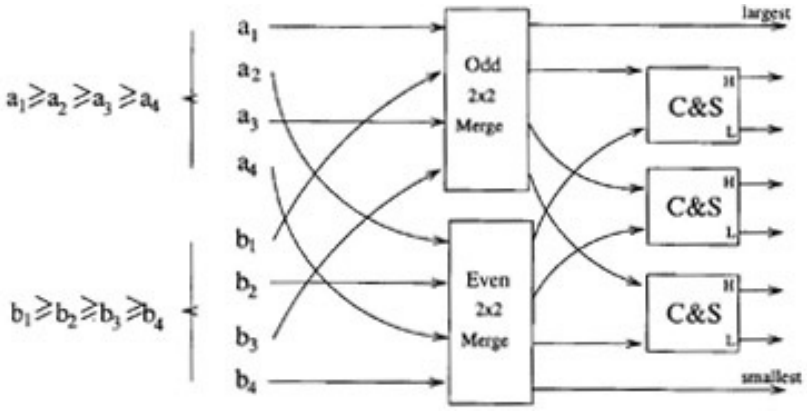

Figure 4-2: A 4×4 merge sort circuit

Other size of merge units can be derived from the 4×4 merge-sort structure. For example, A 4×2 merge-sort structure can be designed by simply treating the last 2 data elements of 4×4 merge-sort architecture as dummy and throwing away any C&S unit that has at least one of those dummy elements as inputs. A 4×2 merge-sort structure requires 6 C&S units.

4.2 Rank-order Filter(ROF) & Parallel ROF

We can start to design the Parallel programmable ROF based on the Batcher's odd-Even Merge-Sort Algorithm. Notice that the inputs to the filters are tapped off a delay line to form a window for the sorter. Figure 4-3 shows a rank order filter structure with window size W=5.

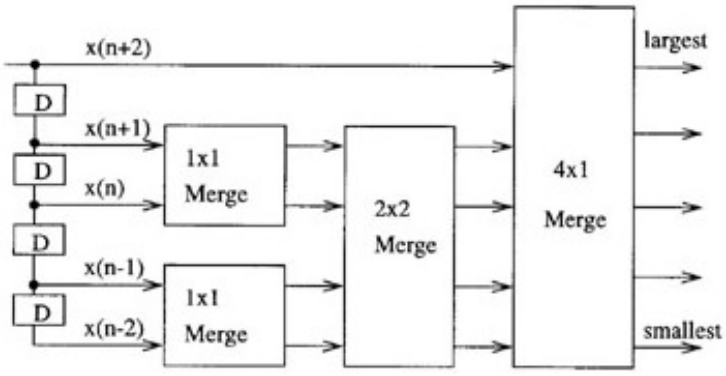

Figure 4-4: W=5, rank order filter.

We can try to exploit substructure sharing technique to design fewest number of compare-swap (**C&S**) unit ROF, and find pairs of inputs common to multiple outputs to decrease the hardware complexity. By applying parallel structure, we can share the comparisons within a block.

Repeating the same hardware twice, and eliminate redundant computations. We can get an efficient ROF structure. This process is shown in the following two figures.

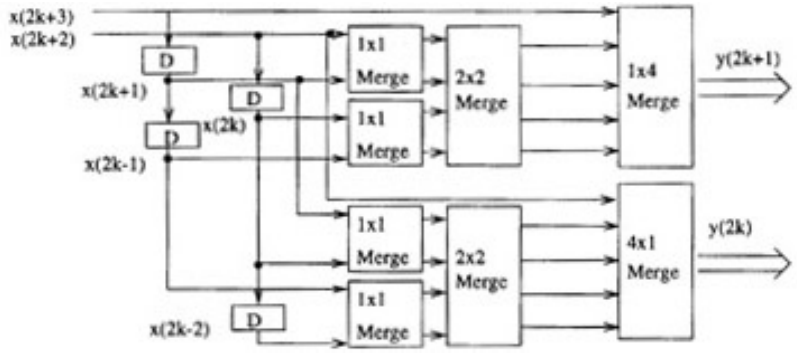

Figure 4-5: W=5, rank order filter architecture with 2 level of parallel processing

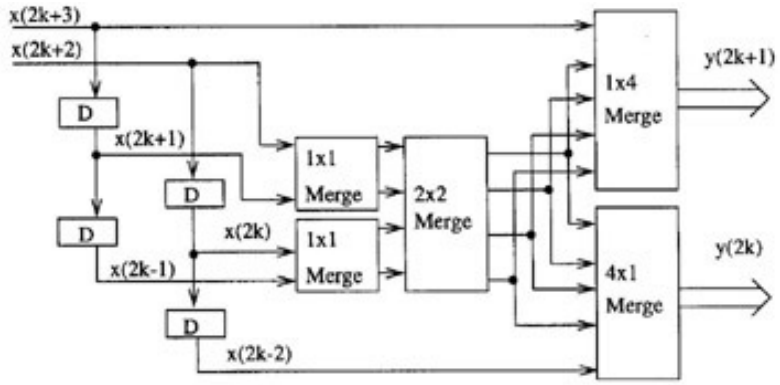

Figure 4-6: W=5, L=2 rank order filter architecture with substructure sharing.

4.3 Running order merge sorter (Time Mapping)

We can take advantage of the *time relation of the inputs* to further reduce the number of merge unit. We call this technique as running order merge sorter, where those units whose inputs are separated by the same number of time steps are mapped onto a single merge unit by using extra memory units for each level of merging. The merging process is shown in the following two pictures.

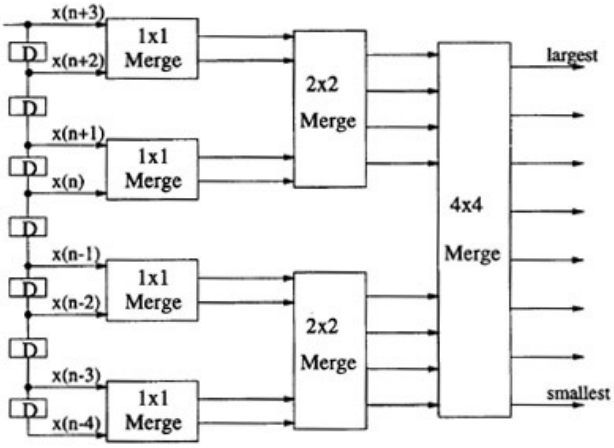

Figure 4-7: Block structure for W=8 rank-order filter

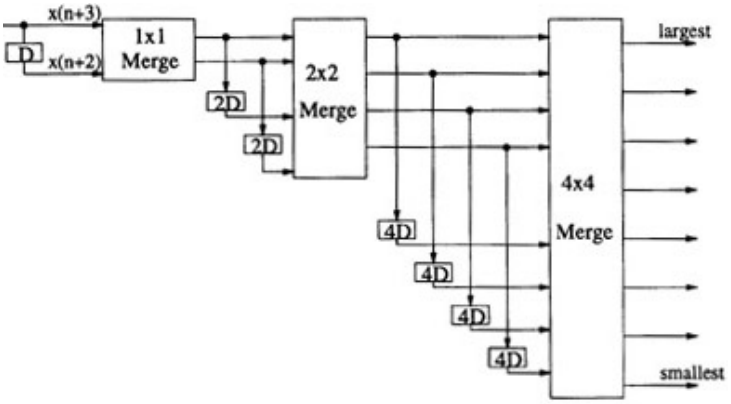

Figure 4-8: Block structure for time mapped W=8 rank-order filter

4.4 Low power rank-order filter

Power consumption reduction using parallel processing is achieved by reducing the supply voltage while maintaining the same sample rate as the sequential systems. However, the reduced the power cannot be lower than the threshold voltage (V_t) of the CMOS device . The parallel system's power consumption can be written as

$$P_{par} = \beta^2 P_{seq} \tag{22}$$

where β is the power reduction factor and βV_0 has to be greater than $2V_t$. Without strength reduction, the area of the parallel system is increased linearly with respect to the block size L. With substructure sharing, we can achieve the power consumption as

$$P_{par} = \alpha\beta^2 P_{seq} \tag{23}$$

where α<1 Hence, it is possible to continue to reduce the power consumption beyond the supply voltage limit by increasing blocksize L.

5. CONCLUSIONS

This report has reviewed the algorithmic strength reduction transformation approaches discussed in the Chapter 9 of [1]. These transformations exploit substructures sharing and reduce the number of stronger operations, possibly at the cost of increasing the number of weaker operations. The application of algorithmic strength reduction in discrete cosine transformation and parallel rank order filters are discussed. These architectures can reduce the area and power consumption in a VLSI implementation or reduce the iteration period in a programmable DSP implementation. The application of algorithmic strength reduction at numerical level to reduce the implementation complexity will be discussed in the following seminar (Chapter 15).

6. REFERENCE

[1] K. Parhi, *VLSI digital signal processing systems: design and implementation.* New York: wiley, 1999.

[2] E. Ifeachor and B. Jervis, *Digital signal processing: A practical approach.* Wokingham: Addison-wesley, 1993.

[3] S. winograd, "arithmetic complexity of computations," in *Proceedings CBMS-NSF regional conference series in applied mathematics,*vol. 3, 1980.

7. PROBLEMS

Problems 1:

Express the 2-parallel filter algorithm:

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

in terms of post-processing matrix, a diagonal matrix, and a pre-processing matrix. Obtain another 2-parallel structure using the transpose of this formulation.

Problems 38

A 3-parallel architecture for a rank-order filter with window size 5 has been derived in Example 9.4.6. Design another 3 parallel architecture by considering y(3k), y(3k+1) and y(3k+2) as a group. Draw detailed circuits for all mergers used in this architecture.