Efficient Architectures for 1-D and 2-D Lifting-Based Wavelet Transforms

Hongyu Liao, Mrinal Kr. Mandal, *Senior Member, IEEE*, and Bruce F. Cockburn, *Member, IEEE*

***Abstract*—The lifting scheme reduces the computational complexity of the discrete wavelet transform (DWT) by factoring the wavelet filters into cascades of simple lifting steps that process the input samples in pairs. We propose four compact and efficient hardware architectures for implementing lifting-based DWTs, namely, one-dimensional (1-D) and two-dimensional (2-D) versions of what we call recursive and dual scan architectures. The 1-D recursive architecture exploits interdependencies among the wavelet coefficients by interleaving, on alternate clock cycles using the same datapath hardware, the calculation of higher order coefficients along with that of the first-stage coefficients. The resulting hardware utilization exceeds 90% in the typical case of a five-stage 1-D DWT operating on 1024 samples. The 1-D dual scan architecture achieves 100% datapath hardware utilization by processing two independent data streams together using shared functional blocks. The recursive and dual scan architectures can be readily extended to the 2-D case. The 2-D recursive architecture is roughly 25% faster than conventional implementations, and it requires a buffer that stores only a few rows of the data array instead of a fixed fraction (typically 25% or more) of the entire array. The 2-D dual scan architecture processes the column and row transforms simultaneously, and the memory buffer size is comparable to existing architectures.**

***Index Terms*—Discrete wavelet transform, dual scan architecture, lifting scheme, recursive architecture.**

I. Introduction

THE advantages of the wavelet transform over conventional transforms, such as the Fourier transform, are now well recognized. In many application areas, the wavelet transform is more efficient at representing signal features that are localized in both time and frequency [1]. Over the past 15 years, wavelet analysis has become a standard technique in such diverse areas as geophysics, meteorology, audio signal processing, and image compression [2]. Significantly, the 2-D biorthogonal discrete wavelet transform (DWT) has been adopted in the recently established JPEG-2000 still image compression standard [3].

Unlike the Fourier transform, the wavelet transform has many possible sets of basis functions [2]. Tradeoffs can be made between the choice of base functions and the complexity of the corresponding hardware implementations. The dyadic wavelet transform can be computed conveniently using Mallat's tree algorithm [4]. As shown in Fig. 1, the forward transform is computed using a series of highpass and lowpass filters, which are denoted by g[-n] and h[-n], respectively, that operate at increasing resolutions along the dimension of the sample index n. The decimated (i.e., downsampled by a factor of two) output of the highpass filters $(d_{j-1}, d_{j-2}, \ldots)$ captures the detail information at each transform stage, that is, at each of the considered resolutions. The decimated output of each lowpass filter $(c_{j-1}, c_{j-2}, \ldots)$ is processed recursively by the lowpass and highpass filters of the next stage. Finally, the decimated output of the lowpass filter of the last stage corresponds to the low-frequency content of the original signal at the lowest considered resolution. In Mallat's algorithm, the inverse transform is calculated using a reverse tree algorithm that repeatedly filters and interleaves the various streams of transform coefficients back into a single reconstructed data sequence.

Much previous work attempted to efficiently implement the DWT using Mallat's algorithm. Lewis *et al.* exploited the characteristics of the Daub-4 wavelet and proposed a multiplierless 2-D DWT architecture [5]. Parhi *et al.* proposed folded and digit-serial architectures [6]. Chakrabarti and Vishwanath presented several architectures implementing the 2-D DWT ranging from SIMD to parallel filter [7]. Grzeszczak *et al.* proposed a systolic array for computing the 1-D DWT [8]. Wu and Chen proposed a 2-D architecture that employs a folding technique [9].

The lifting scheme offers a new way of constructing biorthogonal wavelets [10]. It also provides a more efficient algorithm for calculating classical wavelet transforms. By first factoring a classical wavelet filter into lifting steps, the computational complexity of the corresponding DWT can be reduced by up to 50% [12]. The lifting steps can be easily implemented with ladder-type structures [12], which is different from the direct finite impulse response (FIR) implementations of Mallat's algorithm. Several lifting-based hardware architectures have been proposed recently. The 2-D architecture described by Andra *et al.* is composed of simple processing units and computes one stage of the DWT at a time [13]. Jiang *et al.* proposed a parallel processing architecture that models the DWT computation as a finite state machine and is efficient for computing the wavelet coefficients near the boundary of each segment of the input signal [14]. Lian *et al.* proposed a 1-D folded architecture to improve the hardware utilization [15] for 5/3 and 9/7 wavelet filters. The authors claimed that their folded architecture can achieve 100% utilization. Despite the improvements in the efficiency of the existing architectures, scope remains for further improvements in the hardware utilization. For example,

Manuscript received November 5, 2002; revised May 11, 2003. This work was supported by Micronet R&D under Grant G119990081, by TR*Labs*, and by the Natural Sciences and Engineering Research Council of Canada under Grants G121210634 and OGP0105567. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Xiang-Gen Xia.

The authors are with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 2V4 Canada (e-mail: liao@ee.ualberta.ca; mandal@ee.ualberta.ca; cockburn@ee.ualberta.ca).

Digital Object Identifier 10.1109/TSP.2004.826175

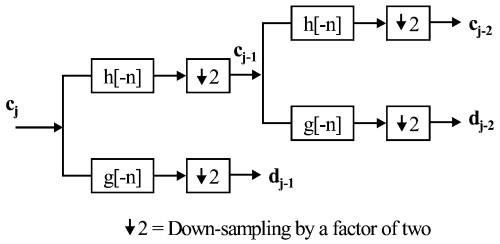


Fig. 1. Block diagram of Mallat's tree algorithm.

the datapath hardware can be kept busier by interleaving either multiple data streams or multiple stages of the DWT calculations on the same datapath.

In this paper, we propose two kinds of lifting-based architectures—dual scan and recursive architectures—to further improve the efficiency of the hardware implementation. The dual scan architecture processes two independent signals (e.g., two rows of an image) simultaneously to achieve 100% hardware utilization. The recursive architecture interleaves the computations of all stages of the DWT into a shared datapath to achieve higher hardware utilization rates.

The rest of the paper is organized as follows: In Section II, the DWT and the lifting scheme are briefly reviewed. The proposed 1-D and 2-D DWT architectures are presented and evaluated in Section III. Concluding remarks are presented in the last section.

II. Discrete Wavelet Transform and Lifting Scheme

In this section, we present a brief review of Mallat's tree algorithm and the lifting scheme. Mallat's algorithm is reviewed in Section II-A. The lifting scheme and the factorization of a wavelet filter are introduced in Sections II-B and C, respectively. The boundary treatment is described in Section II-D.

A. Mallat's Algorithm

The classical DWT can be calculated using an approach known as Mallat's *tree algorithm* [4]. Here, the lower resolution wavelet coefficients of each DWT stage are calculated recursively according to the following equations:

$$c_{j-1,k} = \sum_m c_{j,m} \cdot h[m-2k] \tag{1}$$

$$d_{j-1,k} = \sum_m c_{j,m} \cdot g[m-2k] \tag{2}$$

where

- $c_{p,q}$ qth lowpass coefficient at the pth resolution;
- $d_{p,q}$ qth highpass coefficient at the pth resolution;
- $h[n]$ lowpass wavelet filter corresponding to the mother wavelet;
- $g[n]$ highpass wavelet filter corresponding to the mother wavelet.

The corresponding tree structure for a two-level DWT is illustrated in Fig. 1.

The structure of the corresponding separable 2-D DWT algorithm is shown in Fig. 2, where G and H represent the lowpass and highpass subband filters, respectively. The input image is first decomposed horizontally; the resulting outputs are then decomposed vertically into four subbands usually denoted by LL, LH, HL, and HH. The LL subband can then be further decomposed in the same way.

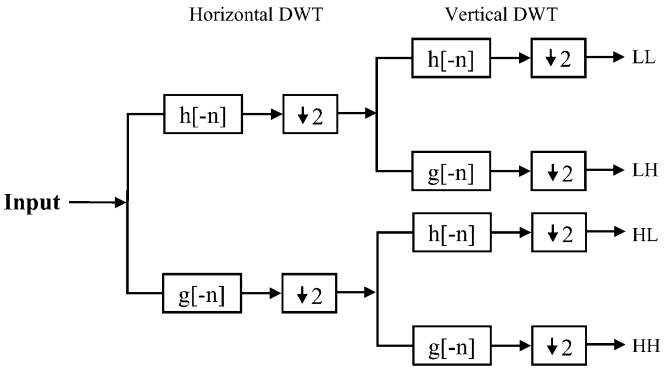


Fig. 2. Block diagram of the 2-D separable DWT.

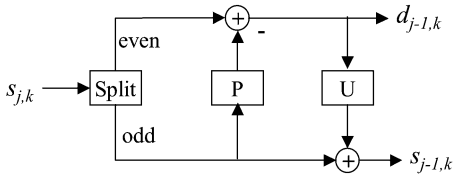


Fig. 3. Lifting scheme.

B. Lifting Scheme

In 1994, Sweldens proposed a more efficient way of constructing the biorthogonal wavelet bases, which he called the lifting scheme [10]. Concurrently, similar ideas were also proposed by Carnicer *et al.* [11]. The basic structure of the lifting scheme is shown in Fig. 3. The input signal $s_{j,k}$ is first split into even and odd samples. The detail (i.e., high-frequency) coefficients $d_{j-1,k}$ of the signal are then generated by subtracting the output of a prediction function P of the odd samples from the even samples. The smooth coefficients (the low-frequency components) are produced by adding the odd samples to the output of an update function U of the details. The computation of either the detail or smooth coefficients is called a *lifting step*.

C. Factoring Wavelet Filters Into Lifting Steps

Daubechies and Sweldens showed that every FIR wavelet or filterbank can be factored into a cascade of lifting steps, that is, as a finite product of upper and lower triangular matrices and a diagonal normalization matrix [12]. The highpass filter $g(z)$ and lowpass filter $h(z)$ in (1) and (2) can thus be rewritten as

$$g(z) = \sum_{i=0}^{J-1} g_i z^{-1} \tag{3}$$

$$h(z) = \sum_{i=0}^{J-1} h_i z^{-1} \tag{4}$$

where J is the filter length. We can split the highpass and lowpass filters into even and odd parts:

$$g(z) = g_e(z^2) + z^{-1} g_o(z^2) \tag{5}$$

$$h(z) = h_e(z^2) + z^{-1} h_o(z^2). \tag{6}$$

The filters can also be expressed as a polyphase matrix as follows:

$$P(z) = \begin{bmatrix} h_e(z) & g_e(z) \\ h_o(z) & g_o(z) \end{bmatrix}. \quad (7)$$

Using the Euclidean algorithm, which recursively finds the greatest common divisors of the even and odd parts of the original filters, the forward transform polyphase matrix $\tilde{P}(z)$ can be factored into lifting steps as follows:

$$\tilde{P}(z) = \prod_{i=1}^{m} \begin{bmatrix} 1 & 0 \\ -s_i(z^{-1}) & 1 \end{bmatrix} \begin{bmatrix} 1 & -t_i(z^{-1}) \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1/K & 0 \\ 0 & K \end{bmatrix}, \quad m \le K. \quad (8)$$

where $s_i(z)$ and $t_i(z)$ are Laurent polynomials corresponding to the update and prediction steps, respectively, and K is a nonzero constant. The inverse DWT is described by the following equation:

$$P(z) = \prod_{i=1}^{m} \begin{bmatrix} 1 & s_i(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ t_i(z) & 1 \end{bmatrix} \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix}. \quad (9)$$

As an example, the lowpass and highpass filters corresponding to the Daubechies 4-tap wavelet can be expressed as [12]

$$\tilde{h}(z) = h_0 + h_1 z^{-1} + h_2 z^{-2} + h_3 z^{-3}$$
$$\tilde{g}(z) = -h_3 z^2 + h_2 z^1 - h_1 + h_0 z^{-1} \quad (10)$$

where

$$h_0 = \frac{1+\sqrt{3}}{4\sqrt{2}}, \quad h_1 = \frac{3+\sqrt{3}}{4\sqrt{2}}, \quad h_2 = \frac{3-\sqrt{3}}{4\sqrt{2}}$$
$$h_3 = \frac{1-\sqrt{3}}{4\sqrt{2}}.$$

Following the above procedure, we can factor the analysis polyphase matrix of the Daubechies-4 wavelet filter as

$$\tilde{P}(z) = \begin{bmatrix} 1 & -\sqrt{3} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \frac{\sqrt{3}}{4} + \frac{\sqrt{3}-2}{4} z^{-1} & 1 \end{bmatrix} \times \begin{bmatrix} 1 & z \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}+1}{\sqrt{2}} & 0 \\ 0 & \left(\frac{\sqrt{3}+1}{\sqrt{2}}\right)^{-1} \end{bmatrix}. \quad (11)$$

The corresponding synthesis polyphase matrix can be factored as

$$P(z) = \begin{bmatrix} \left(\frac{\sqrt{3}+1}{\sqrt{2}}\right)^{-1} & 0 \\ 0 & \frac{\sqrt{3}+1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 & -z \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ -\frac{\sqrt{3}}{4} - \frac{\sqrt{3}-2}{4} z^{-1} & 1 \end{bmatrix} \begin{bmatrix} 1 & \sqrt{3} \\ 0 & 1 \end{bmatrix}. \quad (12)$$

Similarly, the $9/7$ analysis wavelet filter can be factored as [1]:

$$\tilde{P}(z) = \begin{bmatrix} 1 & \alpha(1+z^{-1}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \beta(1+z) & 1 \end{bmatrix} \times \begin{bmatrix} 1 & \gamma(1+z^{-1}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \delta(1+z) & 1 \end{bmatrix} \begin{bmatrix} \zeta & 0 \\ 0 & \zeta^{-1} \end{bmatrix}. \quad (13)$$

The corresponding synthesis wavelet filter is factored as

$$P(z) = \begin{bmatrix} \zeta^{-1} & 0 \\ 0 & \zeta \end{bmatrix} \begin{bmatrix} 1 & -\delta(1+z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -\gamma(1+z^{-1}) & 1 \end{bmatrix} \times \begin{bmatrix} 1 & -\beta(1+z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -\alpha(1+z^{-1}) & 1 \end{bmatrix} \quad (14)$$

where the values of $\alpha, \beta, \gamma, \delta$, and ζ are shown in Fig. 8. The computational cost of calculating two Daub-4 DWT coefficients using (11) is nine operations (five multiplications and four additions). On the other hand, Mallat's algorithm needs 14 operations (eight multiplications and six additions), according to (10). For longer FIR wavelet filters, the speed up can be up to 100% [12], which is a significant improvement for real-time applications.

D. Boundary Treatment

To keep the number of wavelet coefficients the same as the number of data samples in the original signal, an appropriate signal extension method is necessary. Typical signal extension methods are zero padding, periodic extension, and symmetric extension [16]. Zero padding is not normally acceptable for the classical wavelet algorithms due to the extra wavelet coefficients that are introduced. Periodic extension is applicable to all (biorthogonal and orthogonal) wavelet filters, but symmetric extension is suitable only for biorthogonal (symmetric) wavelet filters. Since the lifting scheme applies for constructing biorthogonal wavelets, symmetric extension can always be used to calculate the lifting scheme.

Proposition: Lifting steps obtained by factoring the finite wavelet filter pairs can be calculated by using simple zero padding extension.

Proof Sketch: After a polyphase matrix representing a wavelet transform with finite filters is factored into lifting steps, each step becomes a Laurent polynomial, namely, the $s_i(z)$ or $t_i(z)$ from (8). Since the difference between the degrees of the even and odd parts of a polynomial is never greater than two, we can always find a common divisor of first-order or lower for the polynomials. Hence, a classical wavelet filter can always be factored into first-order or lower order Laurent polynomials (i.e., $s_i(z)$ or $t_i(z)$). Lifting steps containing these short polynomials correspond to one to three-tap FIR filters in the hardware implementations. Because signal extension is not necessary for a two-tap wavelet filter, like for the Haar wavelet, we conclude that zero padding can be used in the lifting algorithm. Q.E.D.

In our work, the easily implemented zero extension is used in the proposed architectures. The sample overlap wavelet transform recommended in JPEG-2000 Part II [3] can also be implemented in the proposed 2-D architecture.

III. Proposed Architectures

To implement the lifting algorithm described above, the input signal has to be first separated into even and odd samples. Each pair of input samples (one even and one odd) is then processed according to the specific analysis polyphase matrix. For many applications, the data can be read no faster than one input sample per clock cycle; therefore, sample pairs are usually processed at every other clock cycle. Hence, this is a limitation on the speed

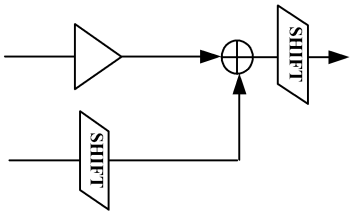


Fig. 4. MAC for asymmetric wavelet filters.

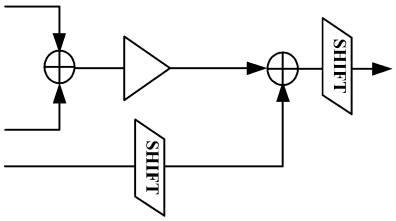


Fig. 5. MAC for symmetric wavelet filters.

and efficiency of a direct implementation of the lifting scheme. To overcome this bottleneck, the proposed recursive architectures exploit the available idle cycles and reuse the same hardware to recursively interleave the DWT stages. The dual scan architectures achieve efficiency gain by keeping the datapath hardware busy with two different streams of data.

A. Proposed 1-D Architectures

1) One–Dimensional Recursive Architecture: Because of the downsampling resulting from the splitting step at each stage in the lifting-based DWT, the number of low-frequency coefficients is always half the number of input samples from the preceding stage. Further, because only the low-frequency DWT coefficients are further decomposed in the dyadic DWT, the total number of the samples to be processed for an L-stage 1-D DWT is

$$N(1+1/2+1/4+\cdots+1/2^{L-1}) = N(2-1/2^{L-1}) < 2N$$

where N is the number of the input samples. For a finite-length input signal, the number of input samples is always greater than the total number of intermediate low-frequency coefficients to be processed at the second and higher stages. Accordingly, there are time slots available to interleave the calculation of the higher stage DWT coefficients while the first-stage coefficients are being calculated.

a) Design Details: The *recursive architecture* (RA) is a general scheme that can be used to implement any wavelet filter that is decomposable into lifting steps [17]. As 1-D examples, we will describe RA implementations of the Daub-4 and 9/7 wavelet filters; in a later section, we will show how the RA can be extended to 2-D wavelet filters. The RA can be extended to even higher dimensions in a similar way, but any example would be too long to include in this article.

The RA is a modular scheme made up of basic circuits such as delay units, pipeline registers, multiplier-accumulators (MACs), and multipliers. Since the factored Laurent polynomials $s_i(z)$ and $t_i(z)$ for symmetric (biorthogonal) wavelet filters are themselves symmetric, and those for asymmetric filters are normally asymmetric, we use two kinds of MACs to minimize the computational cost. The MAC for asymmetric filters (shown in Fig. 4) consist of a multiplier, an adder, and two shifters. The symmetric MAC (shown in Fig. 5) has one more adder than the asymmetric MAC. The shifters are used to scale the partial results so that accuracy can be better preserved.

Different kinds of lifting-based DWT architectures can be constructed by combining the four basic lifting step circuits, which are shown in Fig. 6. The general construction has the following steps.

Step 1) Decompose the given wavelet filter into lifting steps [12].

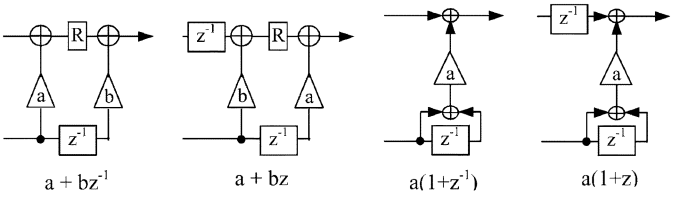
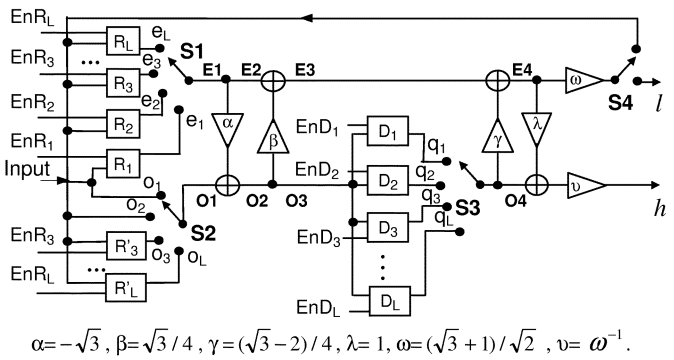


Fig. 6. Circuits for the basic lifting steps.



$\alpha = -\sqrt{3}$, $\beta = \sqrt{3}/4$, $\gamma = (\sqrt{3}-2)/4$, $\lambda = 1$, $\omega = (\sqrt{3}+1)/\sqrt{2}$, $\upsilon = \omega^{-1}$.

Fig. 7. One-dimensional recursive architecture for the Daub-4 DWT. "R" represents registers, and "D" represents delay units. "S_i" represents control signals for the data flow.

Step 2) Construct the corresponding cascade of lifting step circuits. Replace each delay unit in each circuit with an array of delay units. The number of delay units in the array is the same as the number of wavelet stages.

Step 3) At the beginning of the cascade, construct an array of delay units that will be used to split the inputs for all wavelet stages into even and odd samples. These delay units are also used to temporarily delay the samples so that they can be input into the lifting step cascade at the right time slot. Two multiplexer switches are used to select one even input and one odd input to be passed from the delay units to the first lifting step.

Step 4) Construct a data flow table that expresses how all of the switches are set and how the delay units are enabled in each time slot. There is latency as the initial inputs for the first wavelet stage propagate down through the cascade. A free time slot must then be selected to fix the time when the inputs for the second wavelet stage will be sent into the cascade. All higher order stages must also be scheduled into free time slots in the data flow table.

Step 5) Design the control sequencer to implement the data flow table.

The RA in Fig. 7 calculates three stages of the Daub-4 DWT, whereas the RA in Fig. 8 calculates the three-stage 9/7 DWT.

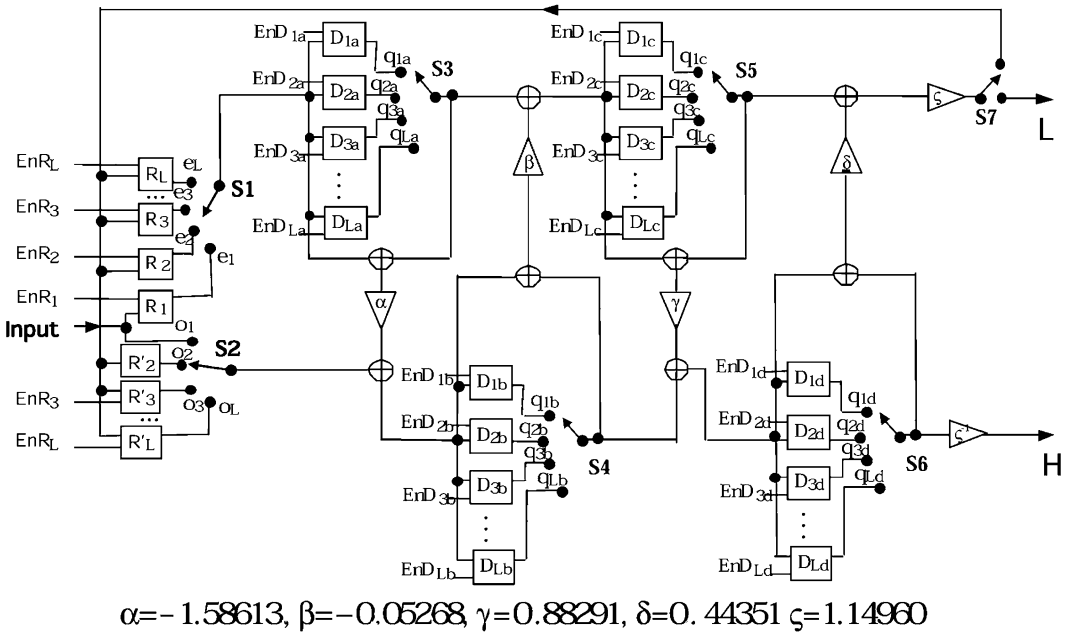


Fig. 8. One-dimensional recursive architecture for the $9/7$ DWT.

Because the control sequence of the RAs for all wavelets is similar, we will discuss the operation of the RA for the Daub-4 DWT in more detail.

In Fig. 7, the input registers R_i $(i = 1, 2, \ldots, L)$ and R'_i $(i = 3, \ldots, L)$ hold the input values for the ith DWT stage. Thus, the first stage coefficients can be calculated at every other clock cycle, and the data for the other stages can be fed into the lifting step pipeline during the intervening cycles. Using $x_{i,j}$ to denote the jth coefficient of the ith stage, the DWT coefficients can be calculated in the order shown in Fig. 9.

The input registers also synchronize the even and odd samples of each stage. Since the first two stages can be immediately processed when the odd samples are ready, no input register is needed for the odd samples for these two stages. Register d_i is a delay unit for the ith stage. After splitting the input data into even and odd parts, the Daub-4 DWT is calculated step by step, as shown in Table I. In Table I, $\mathrm{E_n}$ and $\mathrm{O_n}$ are the outputs of each lifting step; $e_{-i,j}$ and $o_{-i,j}$ denote the even and odd intermediate results of each lifting step. Since the architecture is pipelined by each MAC unit, the outputs of each lifting step are synchronized. As an example, the calculations of the first pair of DWT coefficients are given below:

E1: $x_{0,1} = x_{0,1}$

O1: $x_{0,2} = x_{0,2}$

E2: $e_{-1,1} = x_{0,1}$

O2: $o_{-1,1} = \alpha x_{0,1} + x_{0,2}$

E3: $e_{-1,1} = \beta o_{-1,1} + e_{-1,1}$

O3: $o_{-1,1} = o_{-1,1}$

E4: $e_{-1,1} = z^{-1}\gamma o_{-1,1} + e_{-1,1}$

O4: $o_{-1,1} = z^{-1} o_{-1,1}$

Low-frequency DWT coefficient l: $l_{-1,1} = \omega e_{-1,1}$

High-frequency DWT coefficient h: $h_{-1,1}$
$= \upsilon(\lambda e_{-1,1} + o_{-1,1})$.

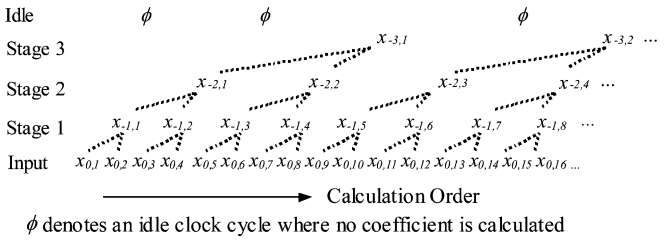


Fig. 9. One-dimensional DWT coefficient computation order.

Therefore, the DWT coefficients of the first stage are generated five clock cycles after the first input sample is received. The first low-frequency DWT coefficient $l_{-1,1}$ is also stored in register R_2. After the second low-frequency DWT coefficient $l_{-1,2}$ is ready, $l_{-1,1}$ and $l_{-1,2}$ are further processed in the idle cycles, as shown in Table I.

The control signals for the switches in a RA can also be deduced from the corresponding data flow table (which is Table I in this case). The timing for the register enable signals is shown in Table II. Switches S1, S2, and S3 steer the data flow at each stage. The timing of the switch control signals is shown in Table III. Output switch S4 feeds back the low-frequency DWT coefficients (except for the last stage) to be further decomposed. The switching timing for S4 is the same as for S1.

The design of the controller is relatively simple, due to the regularity of the control signals for the RA, as shown in Tables II and III. All control signals are generated by counters and flip-flops controlled by a four-state finite state machine. The counters generate periodic signals for the longer period ($T > 4$ clock cycles) control signals, and the flip-flops produce local delays. If externally generated start and stop signals are provided, the long counter for keeping track of the number of input samples is unnecessary. Compared to other direct implementations of lifting-based DWTs, the overhead for the RA controller is very small. The controller should occupy less than 10% of the total silicon area of the 1-D RA.

TABLE I
DATA FLOW FOR THE THREE-STAGE 1-D RECURSIVE ARCHITECTURE

$x_{i,j}$ is input signal, I and j denote the stage and the sequence, respectively; $e_{-I,j}$ and $o_{-I,j}$ are even and odd intermediate results of each lifting step; $l_{-I,j}$ and $h_{-I,j}$ are low and high frequency DWT coefficients

Clk	Input	E_1 ; O_1	E_2 ; O_2	E_3 ; O_3	E_4 ; O_4	l ; h	Stage
1	$x_{0,1}$						
2	$x_{0,2}$	$x_{0,1}$; $x_{0,2}$					
3	$x_{0,3}$		$e_{-1,1}$; $o_{-1,1}$				
4	$x_{0,4}$	$x_{0,3}$; $x_{0,4}$		$e_{-1,1}$; $o_{-1,1}$			
5	$x_{0,5}$		$e_{-1,2}$; $o_{-1,2}$		$e_{-1,1}$; $o_{-1,1}$		
6	$x_{0,6}$	$x_{0,5}$; $x_{0,6}$		$e_{-1,2}$; $o_{-1,2}$		$l_{-1,1}$; $h_{-1,1}$	1
7	$x_{0,7}$		$e_{-1,3}$; $o_{-1,3}$		$e_{-1,2}$; $o_{-1,2}$		
8	$x_{0,8}$	$x_{0,7}$; $x_{0,8}$		$e_{-1,3}$; $o_{-1,3}$		$l_{-1,2}$; $h_{-1,2}$	1
9	$x_{0,9}$	$l_{-1,1}$; $l_{-1,2}$	$e_{-1,4}$; $o_{-1,4}$		$e_{-1,3}$; $o_{-1,3}$		
10	$x_{0,10}$	$x_{0,9}$; $x_{0,10}$	$e_{-2,1}$; $o_{-2,1}$	$e_{-1,4}$; $o_{-1,4}$		$l_{-1,3}$; $h_{-1,3}$	1
11	$x_{0,11}$		$e_{-1,5}$; $o_{-1,5}$	$e_{-1,1}$; $e_{-1,2}$	$e_{-1,4}$; $o_{-1,4}$		
12	$x_{0,12}$	$x_{0,11}$; $x_{0,12}$		$e_{-1,5}$; $o_{-1,5}$	$e_{-1,1}$; $e_{-1,2}$	$l_{-1,4}$; $h_{-1,4}$	1
13	$x_{0,13}$	$l_{-1,3}$; $l_{-1,4}$	$e_{-1,6}$; $o_{-1,6}$		$e_{-1,5}$; $o_{-1,5}$	$l_{-2,1}$; $h_{-2,1}$	2
14	$x_{0,14}$	$x_{0,13}$; $x_{0,14}$	$e_{-2,2}$; $o_{-2,2}$	$e_{-1,6}$; $o_{-1,6}$		$l_{-1,5}$; $h_{-1,5}$	1
15	$x_{0,15}$		$e_{-1,7}$; $o_{-1,7}$	$e_{-2,2}$; $o_{-2,2}$	$e_{-1,6}$; $o_{-1,6}$		
16	$X_{0,16}$	$x_{0,15}$; $x_{0,16}$		$e_{-1,7}$; $o_{-1,7}$	$e_{-2,2}$; $o_{-2,2}$	$l_{-1,6}$; $h_{-1,6}$	1
17	$x_{0,17}$	$l_{-1,5}$; $l_{-1,6}$	$e_{-1,8}$; $o_{-1,8}$		$e_{-1,7}$; $o_{-1,7}$	$l_{-2,2}$; $h_{-2,2}$	2
18	$x_{0,18}$	$x_{0,17}$; $x_{0,18}$	$e_{-2,3}$; $o_{-2,3}$	$e_{-1,8}$; $o_{-1,8}$		$l_{-1,7}$; $h_{-1,7}$	1
19	$x_{0,19}$	$l_{-2,1}$; $l_{-2,2}$	$e_{-1,9}$; $o_{-1,9}$	$e_{-2,3}$; $o_{-2,3}$	$e_{-1,8}$; $o_{-1,8}$		
20	$x_{0,20}$	$x_{0,19}$; $x_{0,20}$	$e_{-3,1}$; $o_{-3,1}$	$e_{-1,9}$; $o_{-1,9}$	$e_{-2,3}$; $o_{-2,3}$	$l_{-1,8}$; $h_{-1,8}$	1
21	$x_{0,21}$	$l_{-1,7}$; $h_{-1,8}$	$e_{-1,10}$; $o_{-1,10}$	$e_{-3,1}$; $o_{-3,1}$	$e_{-1,9}$; $o_{-1,9}$	$l_{-2,3}$; $h_{-2,3}$	2
22	$x_{0,22}$	$x_{0,21}$; $x_{0,22}$	$e_{-2,4}$; $o_{-2,4}$	$e_{-1,10}$; $o_{-1,10}$	$e_{-3,1}$; $o_{-3,1}$	$l_{-1,9}$; $h_{-1,9}$	1
22	$x_{0,22}$	$x_{0,21}$; $x_{0,22}$	$e_{-1,11}$; $o_{-1,11}$	$e_{-2,4}$; $o_{-2,4}$	$e_{-1,10}$; $o_{-1,10}$	$l_{-3,1}$; $h_{-3,1}$	3

TABLE II
ENABLE SIGNALS FOR THE INPUT REGISTERS (k IS THE SAMPLE INDEX) OF THE 1-D RA IMPLEMENTING THE D4 DWT

Time, T_{en} (in clock cycles)	Enable Signals		
$2k$	EnR_1	-	EnD_1 **
$4k+4$	EnR_2	-	EnD_2 **
$8k+9$	EnR_3	EnR'_3 *	EnD_3 **
$2^L k+3\times 2^{L-2}+2^{L-1}-1$	EnR_L	EnR'_L *	EnD_L **

* The actual times are: $T_{en}+2^{L-1}$.
** The actual times are: $T_{en}+2^{L-1}$ + *Latency from S2 to S3*.

TABLE III
INPUT SWITCH CONTROL TIMING FOR THE 1-D RA IMPLEMENTING D4 DWT

Time, T_s (in clock cycles)	Switch Positions		
	S1	S2	S3 *
$2k+1$	e_1	o_1	q_1
$4k+6$	e_2	o_2	q_2
$8k+16$	e_3	o_3	q_3
$2^L k+3\times 2^{L-2}+2^L+$ *Latency*	e_L	o_L	q_L

* The actual times are: T_s+2.

The remaining elements of the RA include registers and switches (tristate buffers). Since the area of the switches is negligible compared with the size of the whole architecture, the cost of the registers dominates. To implement an L-stage DWT, the RA uses $(L-1)(M+1)$ more registers than a conventional lifting-based architecture, where M is the number of delay registers. Considering that a conventional architecture needs an extra memory bank to store at least $N/2$ intermediate DWT coefficients, the RA architecture is more area-efficient in most applications, where $(L-1)(M+1) \ll N/2$. The power consumption of the RA is also likely to be lower than that of a conventional architecture because the RA eliminates the memory read/write operations and because all data routing is local. By avoiding the fetching of data from memories and the driving of long wires, the power dissipated by the RA switches is small. Further discussion of implementation details is beyond the scope of this paper, but our preliminary analysis reveals the potential of the RA architecture in small-size and low-power designs [18].

b) Evaluation: Since the pipeline delay for calculating an L-stage DWT is $L \times T_d$ (where T_d is the latency from input to output) and the sampling-interval for each stage computation increases by two cycles for each additional stage (shown in Fig. 9), the clock cycle count T_P for processing an N-sample DWT can be expressed as

$$\begin{aligned} T_P &= N + (L \times T_d) + (1 + 2 + \cdots + 2^{L-2}) \\ &= N + LT_d + 2^{L-1} - 1. \end{aligned}$$

The hardware utilization can be defined as the ratio of the actual computation time to the total processing time, with time expressed in numbers of clock cycles. At each section of the pipeline structure, the actual clock cycle count T_C is the number of sample pairs to be processed.

$$T_C = (N + N(1 - 2^{1-L}))/2.$$

Note that $N(1-2^{1-L})$ is the number of samples being processed at the second or higher stages. The busy time T_B of the corresponding section can be expressed as

$$T_B = T_P - T_d = N + (L-1)T_d + 2^{L-1} - 1.$$

TABLE IV
COMPUTATION TIME AND HARDWARE UTILIZATION FOR 1-D ARCHITECTURES

N: Number of input samples. T_d, T_{delay}: Circuit delay. L: Number of DWT stages

Architecture	Computation Time (clock cycles)	Hardware Utilization
RA	N+T_dL	50% - 90%
Direct implementation	$2N(1-1/2^L)$+$T_{delay}L$	50%
Folded [15]	$2N(1-1/2^L)$+$T_{delay}L$	≈100%

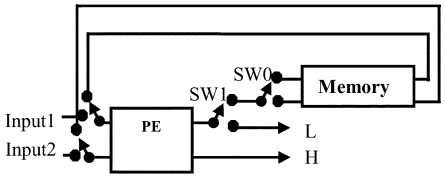


Fig. 10. One-dimensional DSA.

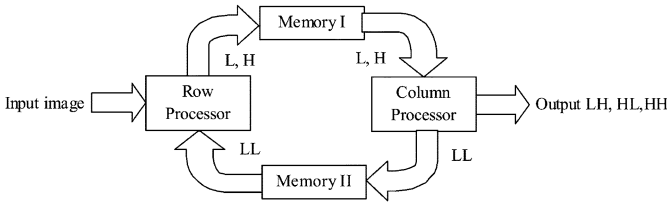


Fig. 11. Conventional 2-D lifting architecture.

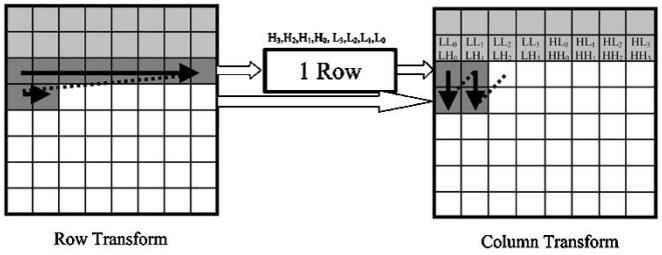


Fig. 12. Calculation sequence of the 2-D RA.

Consequently, the hardware utilization U of the L-stage RA is

$$U = T_C/T_B \times 100\% = \frac{N + N(1-2^{1-L})}{2(N+2^{L-1}+(L-1)T_d-1)} \times 100\%. \quad (15)$$

Because U is a continuous concave function of variable L when $L \geq 1$, the maximum hardware utilization can be achieved when $\partial U/\partial L = 0$. Ignoring the delay $T_d, \partial U/\partial L = 0$ can be expressed as

$$\frac{\partial U}{\partial L} = \frac{N(L-1)2^{-L} - L2^{L-1}}{2(N+2^{L-1}-1)^2} = 0.$$

The above equation is true when $L = 2^{-1}(\log_2 N + \log_2(1 - 1/L) + 1)$. Assuming $L > 1$ and $N \gg \sqrt{N}$, the utilization reaches a maximum of about 90% when $L = 0.5\log_2 N$ and gradually reduces to around 50% when $L = 1$ or $\log_2 N$. For a five-stage DWT operating on 1024 input samples, the utilization approaches 92%. When the number of decomposition stages (L) increases, the processing time increases significantly, and the utilization drops accordingly. As mentioned above, the delay of 2^L is due to the increasing separation (2^L clock cycles) of the input values to each stage. If we decrease the sampling grid for each stage as soon as all previous stages have finished, we can speed up the computation. With little additional controller overhead, the processing time in clock cycle of an L-stage DWT can be reduced to

$$N + (L \times T_d).$$

When $N \to \infty$, the hardware utilization of the 1-D RA approaches 100%. Compared to the conventional implementations of the lifting algorithm, the proposed architectures can achieve a speedup of up to almost 100%, as shown in Table IV.

2) One-Dimensional Dual Scan Architecture: To achieve higher hardware utilization for special cases, we also propose the *dual scan architecture* (DSA), which interleaves the processing of two independent signals simultaneously to increase the hardware utilization. The 1-D DSA is shown in Fig. 10. It consists of a processing element (PE), input and output switches, and two memory units. The PE is a conventional direct hardware implementation of the lifting scheme constructed from the basic building block circuits. The input switches SW are connected to the two input signals when processing the first stage and are connected to the memory when processing the other stages. Switch SW0 separates the low-frequency coefficients of the two input signals. Because the architecture generates one low-frequency coefficient at each clock cycle, SW0 is controlled by the system clock. The output switch SW1 is connected to the output only at the final stage. The size of each of memory unit is $M/2$, where M is the maximum number of input samples.

The 1-D DSA calculates the DWT as the input samples are being shifted in and stores the low-frequency coefficients in the internal memory. When all input samples have been processed, the stored coefficients are retrieved to start computing the next stage.

As the 1-D DSA performs useful calculations in every clock cycle, the hardware utilization for the PE is 100%. The processing time for the L-stage DWT of two N-sample signals is $N + (L \times T_d)$. Compared to conventional implementations for computing two separate signals, the 1-D DSA requires only half the hardware. Hence, given an even number of equal-length signals to process, the speedup of the 1-D DSA is 100%.

B. Proposed 2-D Architectures

A conventional implementation of a separable 2-D lifting-based DWT is illustrated in Fig. 11, where separate row and column processors each use a 1-D lifting architecture. The row processor calculates the DWT of each row of the input image, and the resulting decomposed high- and low-frequency components are stored in memory bank I. Since this bank normally stores all the horizontal DWT coefficients, its size is N^2 for an $N \times N$ image. When the row DWT is completed, the column processor starts calculating the vertical DWT on the coefficients from the horizontally decomposed image. The LH, HL, and HH subbands are final results and can be shifted

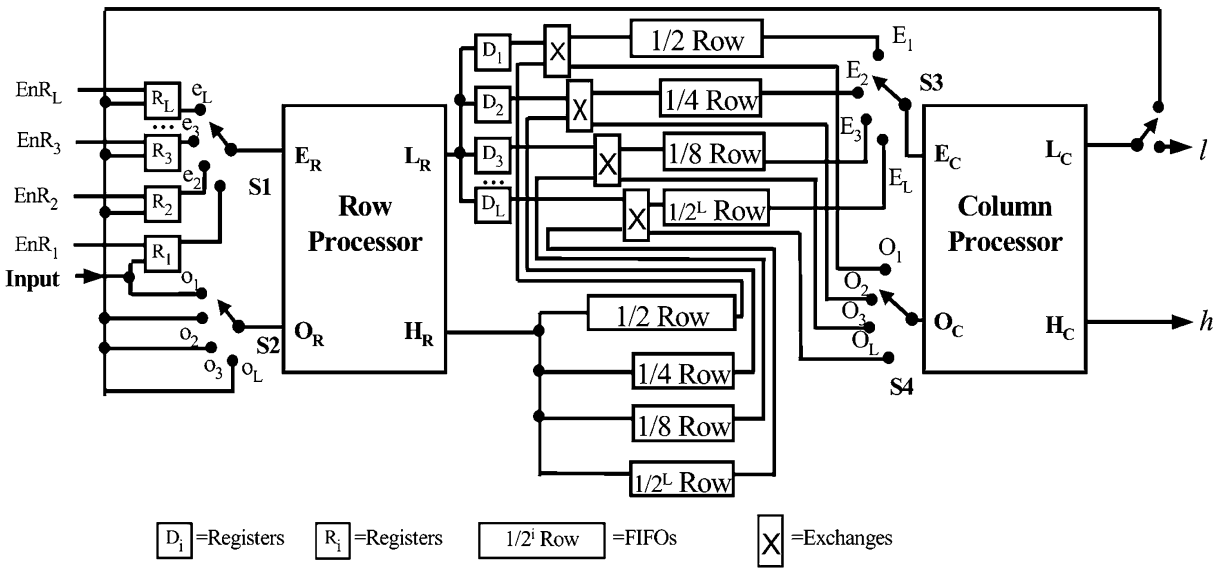


Fig. 13. Two-dimensional RA.

out; the LL subband is stored in memory bank II for further decomposition. The size of memory bank II is thus at least $N^2/4$. Such a straightforward implementation of the 2-D DWT is both time and memory intensive. To increase the computation speed, we propose a 2-D RA and a 2-D DSA for the separable 2-D lifting-based DWT.

1) Two-Dimensional Recursive Architecture: The basic strategy of the 2-D recursive architecture is the same as that of its 1-D counterpart: The calculations of all DWT stages are interleaved to increase the hardware utilization. Within each DWT stage, we use the processing sequence shown in Fig. 12. The image is scanned into the row processor in a raster format, and the first horizontal DWT is immediately started. The resulting high- and low-frequency DWT coefficients of the odd lines are collected and pushed into two first-in first-out FIFO registers or two memory banks. The separate storage of the high- and low-frequency components produces a more regular data flow and reduces the required output switch operations, which in turn consumes less power. The DWT coefficients of the even lines are also rearranged into the same sequence and are directly sent to the column processor, together with the outputs of the FIFO. The column processor starts calculating the vertical DWT in a zigzag format after one row's delay.

A simplified schematic for the 2-D RA is shown in Fig. 13. Note that the row DWT is similar to that of the 1-D DWT so that the datapath of the row processor is the same as for the 1-D RA. The column processor is implemented by replacing the delay registers and input circuit of the 1-D RA with delay FIFOs and the circuitry, as shown in Fig. 13. The interaction between the row and column processor goes as follows: When the row processor is processing the even lines (assuming that it starts with 0th row), the high- and low-frequency DWT coefficients are shifted into their corresponding FIFOs. When the row processor is processing the odd lines, the low-frequency DWT coefficients of the current lines and of the previous lines stored in the FIFOs are sent to the column processor. Registers d_i are used if the low-frequency coefficients are generated before their high-frequency counterparts. At the same time, the high-frequency DWT coefficients of the current lines are shifted into their corresponding FIFOs, and the outputs of these FIFOs are shifted into the FIFOs corresponding to the low frequency. The computations are arranged in such a way that the processings of the DWT coefficients for the first and the other stages can be easily interleaved in neighboring clock cycles. Once the processing of the low-frequency components is done, the outputs of both FIFOs are sent to the column processor. The function of the exchanges, which are shown as X boxes in Fig. 13, is to redirect the data flow between the FIFOs and the input of the column processor. As shown in Fig. 14, the exchange block has two input channels, two output channels, and a control signal. When the control signal SW $= 0$, the data from input channel 1 flows to output channel 1, and the data from input channel 2 flows to output channel 2; when SW $= 1$, one data stream flows from input channel 2 to output channel 1, the other data stream flows from input channel 1 to output channel 2. At the low-frequency output of the column processor, a switch selects the LL subband and sends it back to the row processor for further decomposition.

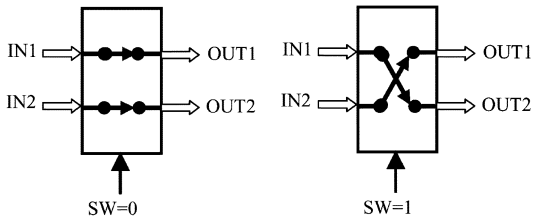


Fig. 14. Exchange operations.

A portion of the data flow for computing an 8×8 sample 2-D Daub-4 DWT is shown in Table V. As described before, the first pair ($e_{-1,1,1}$ and $o_{-1,1,1}$) of the first stage row transform coefficients are generated at the sixth clock cycle. They are immediately shifted into the high- and low-frequency FIFOs, respectively. The consecutive DWT coefficients of the same row

TABLE V
DATA FLOW FOR THE THREE-STAGE 2-D RA

$x_{i,j,k}$ is input signal, *I, j* and *k* denote the stage, the row and column sequences, respectively; $e_{-I,j,k}$ and $o_{-I,j,k}$ are even & odd intermediate results of each lifting step; $l_{-I,j,k}$ and $h_{-I,j,k}$ are low and high frequency DWT coefficients

Clk	Input	Row Processor		FIFOs for Stage 1		Column Processor		Stage
		E_R ; O_R	L_R ; H_R	High Frequency	Low Frequency	E_C ; O_C	Output	
1	$x_{0,1,1}$							
2	$x_{0,2,1}$	$x_{0,1,1}$; $x_{0,2,1}$						
3	$x_{0,3,1}$							
4	$x_{0,4,1}$	$x_{0,3,1}$; $x_{0,4,1}$						
5	$x_{0,5,1}$							
6	$x_{0,6,1}$	$x_{0,5,1}$; $x_{0,6,1}$	$e_{-1,1,1}$; $o_{-1,1,1}$	$e_{-1,1,1}$	$o_{-1,1,1}$			
7	$x_{0,7,1}$							
8	$x_{0,8,1}$	$x_{0,7,1}$; $x_{0,8,1}$	$e_{-1,2,1}$; $o_{-1,2,1}$	$e_{-1,1,1}$; $e_{-1,2,1}$	$o_{-1,1,1}$; $o_{-1,2,1}$			
9	$x_{0,1,2}$							
10	$x_{0,2,2}$	$x_{0,1,2}$; $x_{0,2,2}$	$e_{-1,3,1}$; $o_{-1,3,1}$	$e_{-1,1,1}$; $e_{-1,2,1}$; $e_{-1,3,1}$	$o_{-1,1,1}$; $o_{-1,2,1}$; $o_{-1,3,1}$			
11	$x_{0,3,2}$							
12	$x_{0,4,2}$	$x_{0,3,2}$; $x_{0,4,2}$	$e_{-1,4,1}$; $o_{-1,4,1}$	$e_{-1,1,1}$; $e_{-1,2,1}$; $e_{-1,3,1}$; $e_{-1,4,1}$	$o_{-1,1,1}$; $o_{-1,2,1}$; $o_{-1,3,1}$; $o_{-1,4,1}$			
13	$x_{0,5,2}$							
14	$x_{0,6,2}$	$x_{0,5,2}$; $x_{0,6,2}$	$e_{-1,1,2}$; $o_{-1,1,2}$	$e_{-1,2,1}$; $e_{-1,3,1}$; $e_{-1,4,1}$; $o_{-1,1,1}$	$o_{-1,2,1}$; $o_{-1,3,1}$; $o_{-1,4,1}$; $o_{-1,1,2}$	$e_{-1,1,1}$; $e_{-1,1,2}$		
15	$x_{0,7,2}$							
16	$x_{0,8,2}$	$x_{0,7,2}$; $x_{0,8,2}$	$e_{-1,2,2}$; $o_{-1,2,2}$	$e_{-1,3,1}$; $e_{-1,4,1}$; $o_{-1,1,1}$; $o_{-1,2,1}$	$o_{-1,3,1}$; $o_{-1,4,1}$; $o_{-1,1,2}$; $o_{-1,2,2}$	$e_{-1,2,1}$; $e_{-1,2,2}$		
17	$x_{0,1,3}$							
18	$x_{0,2,3}$	$x_{0,1,3}$; $x_{0,2,3}$	$e_{-1,3,2}$; $o_{-1,3,2}$	$e_{-1,4,1}$; $o_{-1,1,1}$; $o_{-1,2,1}$; $o_{-1,3,1}$	$o_{-1,4,1}$; $o_{-1,1,2}$; $o_{-1,2,2}$; $o_{-1,3,2}$	$e_{-1,3,1}$; $e_{-1,3,2}$	$ll_{-1,1,1}$; $lh_{-1,1,1}$	1
19	$x_{0,3,3}$							
20	$x_{0,4,3}$	$x_{0,3,3}$; $x_{0,4,3}$	$e_{-1,4,2}$; $o_{-1,4,2}$	$o_{-1,1,1}$; $o_{-1,2,1}$; $o_{-1,3,1}$; $o_{-1,4,1}$	$o_{-1,1,2}$; $o_{-1,2,2}$; $o_{-1,3,2}$; $o_{-1,4,2}$	$e_{-1,4,1}$; $e_{-1,4,2}$	$ll_{-1,2,1}$; $lh_{-1,2,1}$	1
21	$x_{0,5,3}$	$ll_{-1,1,1}$; $ll_{-1,2,1}$						
22	$x_{0,6,3}$	$x_{0,5,3}$; $x_{0,6,3}$	$e_{-1,1,3}$; $o_{-1,1,3}$	$o_{-1,2,1}$; $o_{-1,3,1}$; $o_{-1,4,1}$; $e_{-1,1,3}$	$o_{-1,2,2}$; $o_{-1,3,2}$; $o_{-1,4,2}$; $o_{-1,1,3}$	$o_{-1,1,1}$; $o_{-1,1,2}$	$ll_{-1,3,1}$; $lh_{-1,3,1}$	1
22	$x_{0,7,3}$							
23	$x_{0,8,3}$	$x_{0,7,3}$; $x_{0,8,3}$	$e_{-1,2,3}$; $o_{-1,2,3}$	$o_{-1,3,1}$; $o_{-1,4,1}$; $e_{-1,1,3}$; $e_{-1,2,3}$	$o_{-1,3,2}$; $o_{-1,4,2}$; $o_{-1,1,3}$; $o_{-1,2,3}$	$o_{-1,2,1}$; $o_{-1,2,2}$	$ll_{-1,4,1}$; $lh_{-1,4,1}$	1
24	$x_{0,1,4}$	$ll_{-1,3,1}$; $ll_{-1,4,1}$	$e_{-2,1,1}$; $o_{-2,1,1}$					
25	$x_{0,2,4}$	$x_{0,1,4}$; $x_{0,2,4}$	$e_{-1,3,3}$; $o_{-1,3,3}$	$o_{-1,4,1}$; $e_{-1,1,3}$; $e_{-1,2,3}$; $e_{-1,3,3}$	$o_{-1,4,2}$; $o_{-1,1,3}$; $o_{-1,2,3}$; $o_{-1,3,3}$	$o_{-1,3,1}$; $o_{-1,3,2}$	$hl_{-1,1,1}$; $hh_{-1,1,1}$	1
26	$x_{0,3,4}$							
27	$x_{0,4,4}$	$x_{0,3,4}$; $x_{0,4,4}$	$e_{-1,4,3}$; $o_{-1,4,3}$	$e_{-1,1,3}$; $e_{-1,2,3}$; $e_{-1,3,3}$; $e_{-1,4,3}$	$o_{-1,1,3}$; $o_{-1,2,3}$; $o_{-1,3,3}$; $o_{-1,4,3}$	$o_{-1,4,1}$; $o_{-1,4,2}$	$hl_{-1,2,1}$; $hh_{-1,2,1}$	1
28	$x_{0,5,4}$		$e_{-2,2,1}$; $o_{-2,2,1}$					
29	$x_{0,6,4}$	$x_{0,5,4}$; $x_{0,6,4}$	$e_{-1,1,4}$; $o_{-1,1,4}$	$e_{-1,2,3}$; $e_{-1,3,3}$; $e_{-1,4,3}$; $o_{-1,1,3}$	$o_{-1,2,3}$; $o_{-1,3,3}$; $o_{-1,4,3}$; $o_{-1,1,4}$	$e_{-1,1,3}$; $e_{-1,1,4}$	$hl_{-1,3,1}$; $hh_{-1,3,1}$	1
30	$x_{0,7,4}$							
31	$x_{0,8,4}$	$x_{0,7,4}$; $x_{0,8,4}$	$e_{-1,2,4}$; $o_{-1,2,4}$	$e_{-1,3,3}$; $e_{-1,4,3}$; $o_{-1,1,3}$; $o_{-1,2,3}$	$o_{-1,3,3}$; $o_{-1,4,3}$; $o_{-1,1,4}$; $o_{-1,2,4}$	$e_{-1,2,3}$; $e_{-1,2,4}$	$hl_{-1,4,1}$; $hh_{-1,4,1}$	1
32	$x_{0,1,5}$							
33	$x_{0,2,5}$	$x_{0,1,5}$; $x_{0,2,5}$	$e_{-1,3,4}$; $o_{-1,3,4}$	$e_{-1,4,3}$; $o_{-1,1,3}$; $o_{-1,2,3}$; $o_{-1,3,3}$	$o_{-1,4,3}$; $o_{-1,1,4}$; $o_{-1,2,4}$; $o_{-1,3,4}$	$e_{-1,3,3}$; $e_{-1,3,4}$	$ll_{-1,1,2}$; $lh_{-1,1,2}$	1
34	$x_{0,3,5}$							
35	$x_{0,4,5}$	$x_{0,3,5}$; $x_{0,4,5}$	$e_{-1,4,4}$; $o_{-1,4,4}$	$o_{-1,1,3}$; $o_{-1,2,3}$; $o_{-1,3,3}$; $o_{-1,4,3}$	$o_{-1,1,4}$; $o_{-1,2,4}$; $o_{-1,3,4}$; $o_{-1,4,4}$	$e_{-1,4,3}$; $e_{-1,4,4}$	$ll_{-1,2,2}$; $lh_{-1,2,2}$	1
36	$x_{0,5,5}$	$ll_{-1,1,2}$; $ll_{-1,2,2}$						
37	$x_{0,6,5}$	$x_{0,5,5}$; $x_{0,6,5}$	$e_{-1,1,5}$; $o_{-1,1,5}$	$o_{-1,2,3}$; $o_{-1,3,3}$; $o_{-1,4,3}$; $e_{-1,1,5}$	$o_{-1,2,4}$; $o_{-1,3,4}$; $o_{-1,4,4}$; $o_{-1,1,5}$	$o_{-1,1,3}$; $o_{-1,1,4}$	$ll_{-1,3,2}$; $lh_{-1,3,2}$	1
38	$x_{0,7,5}$							
39	$x_{0,8,5}$	$x_{0,7,5}$; $x_{0,8,5}$	$e_{-1,2,5}$; $o_{-1,2,5}$	$o_{-1,3,3}$; $o_{-1,4,3}$; $e_{-1,1,5}$; $e_{-1,2,5}$	$o_{-1,3,4}$; $o_{-1,4,4}$; $o_{-1,1,5}$; $o_{-1,2,5}$	$o_{-1,2,3}$; $o_{-1,2,4}$	$ll_{-1,4,2}$; $lh_{-1,4,2}$	1
40	$x_{0,1,6}$	$ll_{-1,3,2}$; $ll_{-1,4,2}$	$e_{-2,1,2}$; $o_{-2,1,2}$			$e_{-2,1,1}$; $e_{-2,1,2}$		
41	$x_{0,2,6}$	$x_{0,1,6}$; $x_{0,2,6}$	$e_{-1,3,5}$; $o_{-1,3,5}$	$o_{-1,4,3}$; $e_{-1,1,5}$; $e_{-1,2,5}$; $e_{-1,3,5}$	$o_{-1,4,4}$; $o_{-1,1,5}$; $o_{-1,2,5}$; $o_{-1,3,5}$	$o_{-1,3,3}$; $o_{-1,3,4}$	$hl_{-1,1,2}$; $hh_{-1,1,2}$	1
42	$x_{0,3,6}$							
43	$x_{0,4,6}$	$x_{0,3,6}$; $x_{0,4,6}$	$e_{-1,4,5}$; $o_{-1,4,5}$	$e_{-1,1,5}$; $e_{-1,2,5}$; $e_{-1,3,5}$; $e_{-1,4,5}$	$o_{-1,1,5}$; $o_{-1,2,5}$; $o_{-1,3,5}$; $o_{-1,4,5}$	$o_{-1,4,3}$; $o_{-1,4,4}$	$hl_{-1,2,2}$; $hh_{-1,2,2}$	1
44	$x_{0,5,6}$		$e_{-2,2,2}$; $o_{-2,2,2}$			$e_{-2,2,1}$; $e_{-2,2,2}$	$ll_{-2,1,1}$; $lh_{-2,1,1}$	2
45	$x_{0,6,6}$	$x_{0,5,6}$; $x_{0,6,6}$	$e_{-1,1,6}$; $o_{-1,1,6}$	$e_{-1,2,5}$; $e_{-1,3,5}$; $e_{-1,4,5}$; $o_{-1,1,5}$	$o_{-1,2,5}$; $o_{-1,3,5}$; $o_{-1,4,5}$; $o_{-1,1,6}$	$e_{-1,1,5}$; $e_{-1,1,6}$	$hl_{-1,3,2}$; $hh_{-1,3,2}$	1
46	$x_{0,7,6}$							
47	$x_{0,8,6}$	$x_{0,7,6}$; $x_{0,8,6}$	$e_{-1,2,6}$; $o_{-1,2,6}$	$e_{-1,3,5}$; $e_{-1,4,5}$; $o_{-1,1,5}$; $o_{-1,2,5}$	$o_{-1,3,5}$; $o_{-1,4,5}$; $o_{-1,1,6}$; $o_{-1,2,6}$	$e_{-1,2,5}$; $e_{-1,2,6}$	$hl_{-1,4,2}$; $hh_{-1,4,2}$	1
48	$x_{0,1,7}$					$o_{-2,1,1}$; $o_{-2,1,2}$	$ll_{-2,2,1}$; $lh_{-2,2,1}$	2
49	$x_{0,2,7}$	$x_{0,1,7}$; $x_{0,2,7}$	$e_{-1,3,6}$; $o_{-1,3,6}$	$e_{-1,4,5}$; $o_{-1,1,5}$; $o_{-1,2,5}$; $o_{-1,3,5}$	$o_{-1,4,5}$; $o_{-1,1,6}$; $o_{-1,2,6}$; $o_{-1,3,6}$	$e_{-1,3,5}$; $e_{-1,3,6}$	$ll_{-1,1,3}$; $lh_{-1,1,3}$	1
50	$x_{0,3,7}$							
51	$x_{0,4,7}$	$x_{0,3,7}$; $x_{0,4,7}$	$e_{-1,4,6}$; $o_{-1,4,6}$	$o_{-1,1,5}$; $o_{-1,2,5}$; $o_{-1,3,5}$; $o_{-1,4,5}$	$o_{-1,1,6}$; $o_{-1,2,6}$; $o_{-1,3,6}$; $o_{-1,4,6}$	$e_{-1,4,5}$; $e_{-1,4,6}$	$ll_{-1,2,3}$; $lh_{-1,2,3}$	1
52	$x_{0,5,7}$	$ll_{-1,1,3}$; $ll_{-1,2,3}$					$hl_{-2,1,1}$; $hh_{-2,1,1}$	2

are in turn pushed into their conresponding FIFOs in the consequent clock cycles until the end of the row (the 12th clock cycle in this case). When the first pair of the row transform coefficients of the second row is ready, the low-frequency coefficient $(e_{-1,1,2})$ is sent to the odd input of the column processor, and the high-frequency coefficient $o_{-1,1,2})$ is pushed into the corresponding FIFO. The first low-frequency coefficient of the first row $(e_{-1,1,1})$ is also popped out of the FIFO and sent to the even input of the column processor; its high-frequency counterpart $(o_{-1,1,1})$ is pushed to the low-frequency FIFO. After four clock cycles, the column processor generates the first pair of the 2-D DWT coefficients, of which the low-frequency coefficient $(ll_{-1,1,1})$ is temporarily stored in register R_2. The row processor starts further decomposing the low-frequency DWT coefficients after the second low-frequency coefficient $(ll_{-1,2,1})$ is generated (at the 21st clock cycle in Table V).

At the end of the row transform of the second row (at the 20th clock cycle in this case), both FIFOs for the first stage contain only the high-frequency row transform coefficients of the first two rows and start sending these coefficients to the column processor after one clock cycle. As shown in Table V, the calculation of the multiple stage 2-D DWT is continuous and periodic; therefore, control signals for the data flow are easy to generate by relatively simple logic circuits.

Similar to the 1-D RA case, the control signals for the 2-D RA are deduced from the data flow, as shown in Table V. The

TABLE VI
SWITCH CONTROL TIMING FOR THE 2-D RA IMPLEMENTING DAUB-4 DWT

Time, T_s (in clock cycles)	Switch Positions S1	S2	Time, T_s (in clock cycles)	Switch Positions S3 *	S4
$2k+1$	e_1	o_1	$2(l+1)N+2k+6$	E_1	O_1
$2(l+1)N+4k+9$	e_2	o_2	$4(l+1)N+4k+14$	E_2	O_2
$4(l+1)N+8k+17$	e_3	o_3	$8(l+1)N+8k+22$	E_3	O_3
$2^{L-1}(l+1)N+2^L k+1+2T_dL$	e_L	o_L	$2^L(l+1)N+2^L k+3+2T_dL$	E_L	O_L

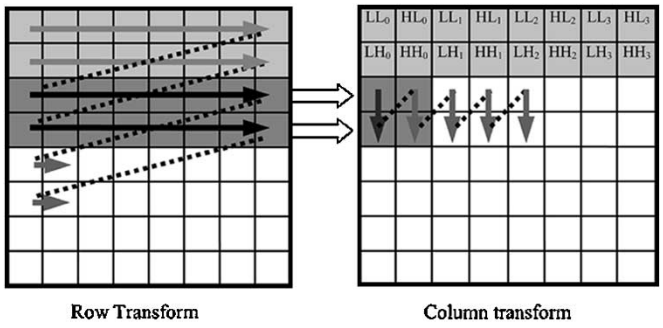


Fig. 15. Scan sequence of the 2-D DSA.

timing for the switch signals of the 2-D RA for lifting-based Daub-4 DWT are shown in Table VI, and the enable signals are fixed delay versions of these switch signals. In addition, similar to the delay reduction method used in the 1-D RA, the delay time of the 2-D DWT can be minimized. The timing of the control signals for other wavelets are similar and can be achieved by changing the delay in Table VI.

Since the high-frequency components are processed one row after the low-frequency components, as shown in Fig. 13 and Table V, the processing delay of the column transform for each stage is roughly one row. In addition, because all the stages are interleaved, the total processing time for an L-stage 2-D DWT is

$$N^2 + N + 2LT_d + 2^{L-1} - 1.$$

Similar to the 1-D implementation, a hardware utilization of about 90% can be achieved when L is close to $\log_2 N$.

2) Two-Dimensional Dual Scan Architecture: In a conventional 2-D DWT algorithm, the vertical DWT is carried out only after the horizontal DWT is finished. This delay between the row and column computations limits the processing speed. The 2-D DSA shortens the delay by adopting a new scan sequence. In applications that can read two pixels per clock cycle from a data buffer, the scan sequence of the 2-D DSA, which is shown in Fig. 15, can be used. The row processor scans along two consecutive rows simultaneously, whereas the column processor also horizontally scans in the row DWT coefficients. In this way, the column processor can start its computation as soon as the first pair of row DWT coefficients is ready. With this improvement, the row and column processors compute the same stage DWT within a few clock cycles of each other.

The structure of the 2-D DSA is shown in Fig. 16. The registers are used to separately hold the even and odd pixels of each row and to interleave the input pairs of each two consecutive rows. The computation timing of the 2-D DSA is shown in Table VII, where the delay of the row and column processor is assumed to be one clock cycle. The row processor of the 2-D DSA is identical to the direct implementation of the 1-D DWT. The column processor is obtained by replacing the 1-pixel delay units in the row processor with 1-row delay units. The low-frequency output switch of the column processor directs the LL subband of each stage DWT to the memory bank. The LL subimage stored in the memory will be returned to the DSA input for further decomposition after the current DWT stage is finished.

The processing time for each stage is $1/2N^2 + 2T_d$. Because only a quarter of the coefficients are further decomposed, the total processing time for an L-stage 2-D DWT is

$$2/3 \times N \times N \times (1 - 1/4^L) + 2 \times T_d \times L.$$

Compared to a conventional implementation, the DSA uses roughly half of the time to compute the 2-D DWT, and the size of the memory for storing the row transform coefficients is reduced to M rows, where M is the number of delay units in a 1-D filter. The comparisons of the processing time and memory size are shown in Tables VIII and IX, respectively. In Table VIII, the timing for the RA is based on one input pixel per clock cycle, whereas the others are based on two input pixels per cycle.

C. Implementation

We implemented the proposed architectures as behavioral-level VHDL models and confirmed their correctness in simulation. As the dynamic range of the DWT coefficients increases with the number of decomposition stages, the number of bits used to represent the coefficients should be large enough to prevent overflow. Bits representing the fractional part can be added to improve the signal-to-noise ratio (SNR) of the calculated DWT coefficients. In our simulation, the filter coefficients and the DWT coefficients are represented in 16 bits (11-bit integer and 5-bit decimal). Therefore, 16-bit multipliers are implemented in our designs, and their results are also rounded to 16 bits. The SNR and PSNR values for the three-stage forward DWT of the test gray-level images are listed in Table X.

The proposed architectures were synthesised and implemented for Xilinx's Virtex II FPGA XC2V250. The 1-D RA implementing the three-stage $9/7$ lifting-based DWT uses 409 logic slices out of the 1536 slices available in the FPGA. The 2-D RA implementing the three-stage Daub-4 DWT uses 879 logic slices and can compute the DWT of 8-bit gray-level images of sizes up to 6000×6000 at 50 MHz using the built-in RAM blocks and multipliers in the FPGA. To estimate the corresponding silicon areas for ASIC designs, we used Synopsys' Design Compiler to synthesize the above architectures with TSMC's 0.18-μm standard cell library aiming for 50-MHz operation. Since the MAC unit is the critical element in the designs, higher operating frequencies can be achieved by implementing faster multipliers or by pipelining the MAC units and minimizing the routing distance of each section of the pipeline. The synthesized designs were then placed and routed by Silicon Ensemble, and the final layouts were generated by using Cadence DFII. The core size of the 1-D RA implementing the three-stage $9/7$ DWT is about 0.177 mm^2 (90% of which is the datapath, 10% is the controller, and the rest is memory),

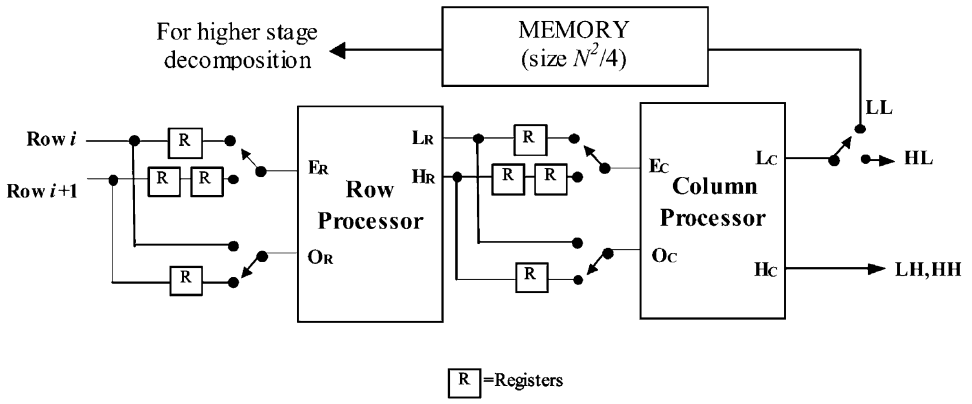


Fig. 16. Two-dimensional DSA.

TABLE VII
DATA FLOW FOR THE 2-D DSA

$x_{i,j}$ is input the signal, I, j the row and column indexes, respectively; $e_{-I,j}$ and $o_{-I,j}$ are even and odd intermediate results of each lifting step; $l_{-I,j}$ and $h_{-I,j}$ are low and high frequency DWT coefficients

Clk	Input	Row Processor		Column Processor	
		E_R ; O_R	L_R ; H_R	E_C ; O_C	L_C ; H_C
1	$x_{1,1}$ $x_{1,2}$				
2	$x_{2,1}$ $x_{2,2}$	$x_{1,1}$; $x_{2,1}$			
3	$x_{3,1}$ $x_{3,2}$	$x_{1,2}$; $x_{2,2}$	$e_{1,1}$; $o_{1,1}$		
4	$x_{4,1}$ $x_{4,2}$	$x_{3,1}$; $x_{4,1}$	$e_{1,2}$; $o_{1,2}$	$e_{1,1}$; $e_{1,2}$	
5		$x_{3,2}$; $x_{4,2}$	$e_{2,1}$; $o_{2,1}$	$o_{1,1}$; $o_{1,2}$	$ll_{1,1}$; $lh_{1,1}$
6			$e_{2,2}$; $o_{2,2}$	$e_{2,1}$; $e_{2,2}$	$hl_{1,1}$; $hh_{1,1}$
7				$o_{2,1}$; $o_{2,2}$	$ll_{2,1}$; $lh_{2,1}$
8					$hl_{2,1}$; $hh_{2,1}$

TABLE VIII
COMPUTATION TIME AND HARDWARE UTILIZATION FOR 2-D ARCHITECTURES

NxN: Size of the input image. T_d, T_{delay}: Circuit delay. L: Number of DWT stages

Architecture (9/7 DWT)	Computation Time (clock cycles)	Hardware Utilization
RA	$N\times N+N+2\times L\times T_d+2^{L-1}$-1	50% - 70%
DS	$2/3\times N\times N\times(1-1/4^L)+2\times T_d\times L$	≈100%
Direct implementation	$4/3\times N\times N\times(1-1/4^L)+2\times T_d\times L$	50%
ACT[13]	$N\times N\times 4/3\times(1-1/4^L)+T_d\times L$	≈ 50%

TABLE IX
COMPARISON OF MEMORY SIZE FOR 2-D ARCHITECTURES

NxN: Size of the input image. T_d, T_{delay}: Circuit delay. L: Number of DWT stages

Architecture	Memory Size
RA for 9/7 wavelet	$4N$
RA for D4 wavelet	$10N$
DSA for 9/7 wavelet	$N\times N/4+4N$
Direct implementation	$5/4\times N\times N$
ACT[13] for 9/7 wavelet	$\cong N\times N/4$ (external memory)

TABLE X
SNR/PSNR VALUES FOR THREE-STAGE FORWARD DWT

	Lena		Barbara	
	SNR	PSNR	SNR	PSNR
Daub-4	69.6529	75.32	69.2755	75.18
9/7	69.1437	74.85	68.7880	74.73

and the core size of the 2-D RA that calculates the three-stage Daub-4 DWT of a 256×256 image is about 2.25 mm^2 (about 15% of which is the datapath, 5% is the controller, and the rest is memory). The core area could be reduced by reimplementing the delay units as register files instead of separate flip-flops, and the performance of the proposed arhcitectures can be further improved by optimizing the circuit designs.

IV. CONCLUSION

In this paper, we proposed two recursive architectures and two dual scan architectures for computing the DWT based on the lifting scheme. Compared to previous implementations of the lifting-based DWT, the proposed architectures have higher hardware utilization and shorter computation time. In addition, since the recursive architectures can continuously compute the DWT coefficients as soon as the samples become available, the memory size required for storing the intermediate results is minimized. Hence, the sizes and power consumptions of both the 1-D and 2-D recursive architectures are significantly reduced, compared with other implementations. In addition, since the designs are modular, they can be easily extended to implement any separable multidimensional DWT by cascading the basic 1-D DWT processors.

Multiwavelet analysis has been found promising in many applications, such as image compression and denoising. Recent research has revealed that multiwavelets can also be constructed by the lifting scheme, and any compactly supported multiwavelet can be factored into lifting steps [19], [20]. As part of future work, we plan to investigate the possibility of extending our architectures to implement the lifting scheme for multiwavelets.

REFERENCES

[1] I. Daubechies, "The wavelet transform time-frequency localization and signal analysis," *IEEE Trans. Inform. Theory*, vol. 36, pp. 961–1004, Sept. 1990.

[2] C. Torrence and G. P. Compo, "A practical guide to wavelet analysis," *Bull. Amer. Meteorological Soc.*, vol. 79, no. 1, pp. 61–78, Jan. 1998.

[3] JPEG2000 Part II Final Committee Draft (2000, Dec.). [Online]. Available: http://www.jpeg.org

[4] S. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 11, pp. 674–693, July 1989.

[5] A. Lewis and G. Knowles, "VLSI architecture for 2-D Daubechies wavelet transform without multipliers," *Electron. Lett.*, vol. 27, no. 2, pp. 171–173, Jan. 1991.

[6] K. Parhi and T. Nishitani, "VLSI architectures for discrete wavelet transforms," *IEEE Trans. VLSI Syst.*, vol. 1, pp. 191–202, June 1993.

[7] C. Chakrabarti and M. Vishwanath, "Efficient realizations of the discrete and continuous wavelet transforms: From single chip implementation to mappings on SIMD array computers," *IEEE Trans. Signal Processing*, vol. 43, pp. 759–769, Mar. 1995.

[8] A. Grzeszczak, M. K. Mandal, S. Panchanathan, and T. Yeap, "VLSI implementation of discrete wavelet transform," *IEEE Trans. VLSI Syst.*, vol. 4, pp. 421–433, Dec. 1996.

[9] P. Wu and L. Chen, "An efficient architecture for two-dimensional discrete wavelet transform," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, pp. 536–544, Apr. 2001.

[10] W. Sweldens, "The lifting scheme: A new philosophy in biorthogonal wavelet construction," *J. Fourier Anal. Appl.*, vol. 4, pp. 247–269, 1998.

[11] J. M. Carnicer, W. Dahmen, and J. M. Pena, "Local decomposition of refinable spaces," *Appl. Comput. Harm. Anal.*, vol. 3, pp. 127–153, 1996.

[12] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps," *J. Fourier Anal. Appl.*, vol. 4, no. 3, pp. 245–267, 1998.

[13] K. Andra, C. Chakrabarti, and T. Acharya, "A VLSI architecture for lifting-based forward and inverse wavelet transform," *IEEE Trans. Signal Processing*, vol. 50, pp. 966–977, Apr. 2002.

[14] W. Jiang and A. Ortega, "Parallel architecture for the discrete wavelet transform based on the lifting factorization," in *Proc. SPIE Conf. Parallel Distributed Methods Image Process. III*, vol. 3817, Denver, CO, July 1999, pp. 2–13.

[15] C. Lian, K. Chen, H. Chen, and L. Chen, "Lifting based discrete wavelet transform architecture for JPEG2000," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2, Sydney, Australia, May 2001, pp. 445–448.

[16] C. M. Brislawn, "Classification of nonexpansive symmetric extension transforms for multirate filter banks," *Appl. Comput. Harmon. Anal.*, vol. 3, pp. 337–357, 1996.

[17] H.-Y. Liao, M. K. Mandal, and B. F. Cockburn, "Efficient implementation of the lifting-based discrete wavelet transform," *Electron. Lett.*, vol. 38, no. 18, pp. 1010–1012, Aug. 29, 2002.

[18] H.-Y. Liao, "Efficient architectures for the lifting-based wavelet transform," M.Sc. Thesis, Dept. Elect. Comput. Eng., Univ. Alberta, , Edmonton, AB, Canada, Spring 2004.

[19] G. Davis, V. Strela, and R. Turcajova, "Multiwavelet construction via the lifting scheme," in *Wavelet Analysis and Multiresolution Methods*, T. X. He, Ed. New York: Marcel Dekker, 2000.

[20] S. Goh, Q. Jiang, and T. Xia, "Construction of biorthogonal multiwavelets using the lifting scheme," *Appl. Comput. Harmon. Anal.*, vol. 9, no. 3, pp. 336–352, Nov. 2000.

Hongyu Liao received the B.Eng. degree in automatic control from Xidian University, Xi'an, China, in 1992. He is presently pursuing the M.Sc. degree at the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada.

From 1992 to 2000, he held several engineering positions in China and Canada. His research interests include efficient implementations of wavelet algorithms, multimedia hardware, and VLSI design.

Mrinal Kr. Mandal (M'99–SM'03) received the M.A.Sc. and Ph.D. degrees in electrical and computer engineering from the University of Ottawa, Ottawa, ON, Canada.

He worked as a scientist in the Indian Space Research Organization, Ahmedabad, India, and is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada. His current research interests include image and video processing, storage and retrieval of images and video, wavelets and filterbanks, VLSI architecture, and network optimization. He has published over 50 papers in refereed journals and conferences. He is also the author of the book *Multimedia Signals and Systems* (Boston, MA, Kluwer, 2002). He was the Principal Investigator (PI) of projects funded by the Canadian Institute for Telecommunication Research (CITR) and the Canadian Networks of Centres of Excellence on Microelectronics (Micronet R&D). He is currently a PI of a project funded by the Natural Sciences and Engineering Research Council of Canada (NSERC).

Dr. Mandal is a Member of SPIE, ACM, and IEICE.

Bruce F. Cockburn (M'90) received the B.Sc. degree in engineering physics from Queen's University, Kingston, ON, Canada, in 1981. He received the M.Math. and Ph.D. degrees in computer science from the University of Waterloo, Waterloo, ON, in 1985 and 1990, respectively.

From 1981 to 1983, he was a Test Engineer and Software Designer at Mitel Corporation, Kanata, ON. Since 1990, he has held an academic appointment with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada, where he is presently an Associate Professor. He is also a research scientist at TR*Labs*, Edmonton. His research interests include VLSI design and test, high-performance parallel signal processing, applications of FGPA technology, logic-enhanced memory architecture, and novel semiconductor memory technologies.

Dr. Cockburn is a member of the ACM and is a registered Professional Engineer in the province of Alberta.