Journal of VLSI Signal Processing 42, 321–339, 2006
© 2006 Springer Science + Business Media, Inc. Manufactured in The Netherlands.
DOI: 10.1007/s11266-006-4191-3

A Survey on Lifting-based Discrete Wavelet Transform Architectures

TINKU ACHARYA AND CHAITALI CHAKRABARTI
Department of Electrical Engineering, Arizona State University, Tempe, Arizona 85287-5706

Received July 29, 2004; Revised June 14, 2005; Accepted August 4, 2005

Published online: 13 February 2006

Abstract. In this paper, we review recent developments in VLSI architectures and algorithms for efficient implementation of lifting based Discrete Wavelet Transform (DWT). The basic principle behind the lifting based scheme is to decompose the finite impulse response (FIR) filters in wavelet transform into a finite sequence of simple filtering steps. Lifting based DWT implementations have many advantages, and have recently been proposed for the JPEG2000 standard for image compression. Consequently, this has become an area of active research and several architectures have been proposed in recent years. In this paper, we provide a survey of these architectures for both 1-dimensional and 2-dimensional DWT. The architectures are representative of many design styles and range from highly parallel architectures to DSP-based architectures to folded architectures. We provide a systematic derivation of these architectures along with an analysis of their hardware and timing complexities.

Keywords: architecture, Discrete Wavelet Transform, lifting, VLSI

1. Introduction

The Discrete Wavelet Transform (DWT) has become a very versatile signal processing tool over the last decade. In fact, it has been effectively used in signal and image processing applications ever since Mallat [1] proposed the multiresolution representation of signals based on wavelet decomposition. The advantage of DWT over other traditional transformations is that it performs multiresolution analysis of signals with localization both in time and frequency. The DWT is being increasingly used for image compression today since it supports features like progressive image transmission (by quality, by resolution), ease of compressed image manipulation, region of interest coding, etc. In fact, it is the basis of the new JPEG2000 image compression standard which has been shown to have superior performance compared to the current JPEG standard [2].

DWT has traditionally been implemented by convolution or FIR filter bank structures. Such implementations require both a large number of arithmetic computations and a large storage—features that are not desirable for either high speed or low power image/video processing applications. Recently, a new mathematical formulation for wavelet transformation has been proposed by Swelden [3] based on spatial construction of the wavelets and a very versatile scheme for its factorization has been suggested in [4]. This new approach is called the lifting-based wavelet transform or simply lifting. The main feature of the lifting-based DWT scheme is to break up the high-pass and low-pass wavelet filters into a sequence of upper and lower triangular matrices, and convert the filter implementation into banded matrix multiplications [4]. This scheme often requires far fewer computations compared to the convolution based DWT [3, 4] and offers many other advantages, as described later in Section 2.

The popularity of lifting-based DWT has triggered the development of several architectures in recent years. These architectures range from highly parallel architectures to programmable DSP-based architectures to folded architectures. In this paper we present a survey of these architectures. We provide a systematic derivation of these architectures and comment on their hardware and timing requirements.

The rest of the paper is organized as follows. In Section 2, we briefly explain the mathematical formulation and principles behind the lifting scheme. In Section 3, we present a number of one-dimensional lifting-based DWT architectures suitable for VLSI implementation. Specifically, we describe direct mapping of the data dependency diagram of the lifting scheme in a pipelined architecture, its variants for improved performance, programmable architectures and implementation of lifting in DSP and recursive architectures. We also present a comparison of the hardware and timing complexities of all the architectures. In Section 4, we present the memory configuration for 2-dimensional DWT architectures, followed by descriptions of a few representative architectures and a comparison of their hardware and timing complexities. We conclude this paper in Section 5.

2. DWT and Lifting Implementation

In traditional convolution (filtering) based approach for computation of the forward DWT, the input signal (x) is filtered separately by a low-pass filter ($\tilde{h}$) and a high-pass filter ($\tilde{g}$). The two output streams are then sub-sampled by simply dropping the alternate output samples in each stream to produce the low-pass (y_L) and high-pass (y_H) subband outputs as shown in Fig. 1. The two filters ($\tilde{h}$, $\tilde{g}$) form the *analysis* filter bank. The original signal can be reconstructed by a *synthesis* filter bank (h, g) starting from y_L and y_H as shown in Fig. 1. Given a discrete signal $x(n)$, the output signals $y_L(n)$ and $y_H(n)$ in Fig. 1 can be computed as follows:

$$y_L(n) = \sum_{i=0}^{T_L-1} \tilde{h}(i)x(2n-i),$$
$$y_H(n) = \sum_{i=0}^{T_H-1} \tilde{g}(i)x(2n-i) \tag{1}$$

where τ_L and τ_H are the lengths of the low-pass ($\tilde{h}$) and high-pass ($\tilde{g}$) filters respectively. During the inverse transform computation, both y_L and y_H are first up-

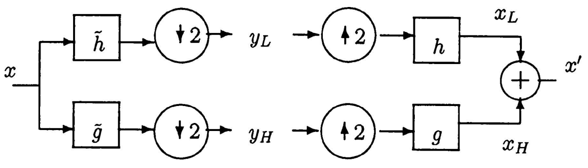


Figure 1. Signal analysis and reconstruction in 1D DWT.

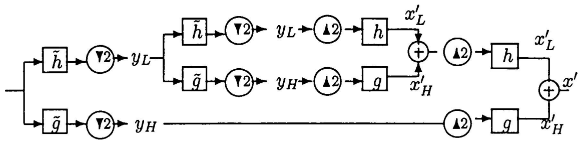


Figure 2. Signal analysis and reconstruction in two-level 1D DWT.

sampled by inserting zeros in between two samples and then filtered by low-pass (h) and high-pass (g) filters respectively. Then they are added together to obtain the reconstructed signal (x') as shown in Fig. 1.

For multiresolution wavelet decomposition, the low-pass subband (y_L) is further decomposed in a similar fashion in order to get the second-level of decomposition, and the process repeated. The inverse process follows similar multi-level synthesis filtering in order to reconstruct the signal. A two level DWT decomposition and its reconstruction have been shown in Fig. 2, as an example. Since two dimensional wavelet filters are separable functions, 2D DWT can be obtianed by first applying the 1D DWT row-wise (to produce L and H subbands in each row) and then column-wise as shown in Fig. 3(a). In the first level of decomposition, four subbands LL1, LH1, HL1 and HH1 are obtained. Repeating the same in the LL1 subband, it produces LL2, LH2, HL2 and HH2 and so on, as shown in Fig. 3(c).

For the filter bank in Fig. 1, the conditions for perfect reconstruction of a signal [4] are given by

$$\begin{aligned} h(z)\tilde{h}(z^{-1}) + g(z)\tilde{g}(z^{-1}) &= 2, \\ h(z)\tilde{h}(-z^{-1}) + g(z)\tilde{g}(-z^{-1}) &= 0 \end{aligned} \tag{2}$$

where $h(z)$ is the Z-transform of the FIR filter h. h can be expressed as a Laurent polynomial of degree p as

$$h(z) = \sum_{i=0}^{p} h_i z^{-i}$$

which can also be expressed using a polyphase representation as

$$h(z) = h_e(z^2) + z^{-1}h_o(z^2) \tag{3}$$

where h_e contains the even coefficients and h_o contains the odd coefficients of the FIR filter h. Similarly,

$$\begin{aligned} g(z) &= g_e(z^2) + z^{-1}g_o(z^2), \\ \tilde{h}(z) &= \tilde{h}_e(z^2) + z^{-1}\tilde{h}_o(z^2), \\ \tilde{g}(z) &= \tilde{g}_e(z^2) + z^{-1}\tilde{g}_o(z^2) \end{aligned} \tag{4}$$

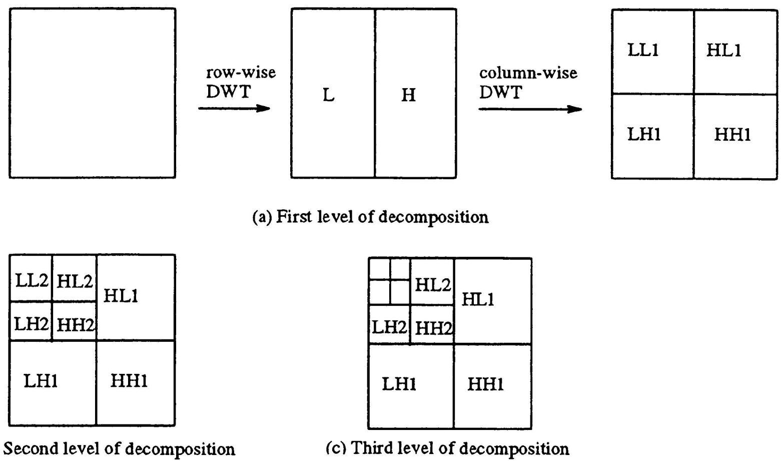

row-wise DWT
L
H
column-wise DWT
LL1
HL1
LH1
HH1
(a) First level of decomposition
LL2
HL2
LH2
HH2
(b) Second level of decomposition
(c) Third level of decomposition

Figure 3. Three levels of decomposition in 2D DWT.

Based on the above formulation, we can define the *polyphase matrices* as

$$\tilde{P}(z) = \begin{bmatrix} \tilde{h}_e(z) & \tilde{h}_o(z) \\ \tilde{g}_e(z) & \tilde{g}_o(z) \end{bmatrix}, \quad P(z) = \begin{bmatrix} h_e(z) & g_e(z) \\ h_o(z) & g_o(z) \end{bmatrix} \tag{5}$$

Often $P(z)$ is called the *dual* of $\tilde{P}(z)$ and for perfect reconstruction, they are related as $P(z)\tilde{P}(z^{-1})^T = I$, where I is the 2×2 *identity* matrix. Now the wavelet transform in terms of the polyphase matrix can be expressed as

$$\begin{bmatrix} y_L(z) \\ y_H(z) \end{bmatrix} = \tilde{P}(z) \begin{bmatrix} x_e(z) \\ z^{-1}x_o(z) \end{bmatrix}, \quad \begin{bmatrix} x_e(z) \\ z^{-1}x_o(z) \end{bmatrix} = P(z) \begin{bmatrix} y_L(z) \\ y_H(z) \end{bmatrix}$$

for the forward DWT and inverse DWT respectively. If the determinant of $P(z)$ is unity, it can be shown by applying Cramer's rule [4] that

$$\tilde{h}(z) = -z^{-1}g(-z^{-1}), \quad \tilde{g}(z) = z^{-1}h(-z^{-1})$$

and hence

$$h(z) = -z^{-1}\tilde{g}(-z^{-1}), \quad g(z) = z^{-1}\tilde{h}(-z^{-1}).$$

When the determinant of $P(z)$ is unity, the synthesis filter pair (h, g) and the analysis filter pair $(\tilde{h}, \tilde{g})$, are both *complementary*. When $(h, g) = (\tilde{h}, \tilde{g})$, the wavelet transformation is called orthogonal, otherwise it is biorthogonal.

It has been shown in [3, 4] that if $(\tilde{h}, \tilde{g})$ is a complementary filter pair, we can apply the Euclidean algorithm to factorize $\tilde{P}(z)$ into a finite sequence of alternating upper and lower triangular matrices as follows:

$$\tilde{P}(z) = \left\{ \prod_{i=1}^{m} \begin{bmatrix} 1 & \tilde{s}_2(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \tilde{t}_i(z) & 1 \end{bmatrix} \right\} \begin{bmatrix} K & 0 \\ 0 & \frac{1}{K} \end{bmatrix} \tag{6}$$

where K is a constant and act as a scaling factor (so is $\frac{1}{K}$), $\tilde{s}_i(z)$ and $\tilde{t}_i(z)$ (for $1 \le i \le m$) are Laurent polynomials of lower orders. Computation of the upper triangular matrix is known as *primal lifting* and this is referred to in the literature as lifting the low-pass subband with the help of the high-pass subband [3, 4]. Similarly, computation of the lower triangular matrix is called *dual lifting*, which is lifting the high-pass subband with the help of the low-pass subband [3, 4]. Often these two basic lifting steps are called *update* and *predict* as well. The dual polyphase factorization which also consists of predict and update steps can be represented in the following form:

$$P(z) = \left\{ \prod_{i=1}^{m} \begin{bmatrix} 1 & 0 \\ -t_i(z^{-1}) & 1 \end{bmatrix} \begin{bmatrix} 1 & -s_i(z^{-1}) \\ 0 & 1 \end{bmatrix} \right\} \begin{bmatrix} \frac{1}{k} & 0 \\ 0 & k \end{bmatrix} \tag{7}$$

Hence the lifting based forward wavelet transform essentially is to first apply the *lazy* wavelet on the input stream (split into even and odd samples), then alternately execute *primal* and *dual* lifting steps, and finally *scale* the two output streams by $\frac{1}{K}$ and K respectively, to produce low-pass and high-pass subbands, as shown

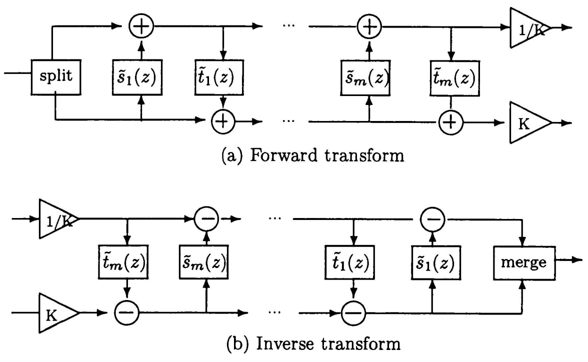


Figure 4. Lifting based forward and inverse DWT.

in Fig. 4(a). The inverse DWT can be derived by traversing above steps in the reverse direction, first scaling the low-pass and high-pass subband inputs by K and $\frac{1}{K}$ respectively, and then applying the dual and primal lifting steps after reversing the signs of coefficients in $\tilde{t}(z)$ and $\tilde{s}(z)$ and finally the inverse lazy transform by up-scaling the output before merging them into a single reconstructed stream as shown in Fig. 4(b).

Due to the linearity of the lifting scheme, if the input data is in integer format, it is possible to maintain data to be in integer format throughout the transform by introducing a rounding function in the filtering operation. Due to this property, the transform is reversible (i.e. lossless) and is called Integer Wavelet Transform (IWT) [5]. It should be noted that filter coefficients need not be integers for IWT. However, if a scaling step is present in the factorization, IWT cannot be achieved. It has been proposed in [5] to split the scaling step into additional lifting steps to achieve IWT.

Example. Consider the (5, 3) filter that has been used in JPEG2000 standard, with $\tilde{h} = (-\frac{1}{8}, \frac{1}{4}, \frac{3}{4}, \frac{1}{4}, -\frac{1}{8})$ and $\tilde{g} = (-\frac{1}{2}, 1, -\frac{1}{2})$.

$$\tilde{h}(z) = -\frac{1}{8}z^{-2} + \frac{1}{4}z^{-1} + \frac{3}{4}z^{0} + \frac{1}{4}z - \frac{1}{8}z^{2},$$
$$\tilde{g}(z) = -\frac{1}{2}z^{-2} + z^{-1} - \frac{1}{2}z^{0}$$

From above equations, we can easily derive that

$$\tilde{h}_e(z^2) = -\frac{1}{8}z^{-2} + \frac{3}{4} - \frac{1}{8}z^2, \quad \tilde{h}_o(z^2) = \frac{1}{4} + \frac{1}{4}(z^2),$$
$$\tilde{g}_e(z^2) = -\frac{1}{2}z^{-2} - \frac{1}{2}, \quad \tilde{g}_o(z^2) = 1.$$

As a result, polyphase matrix of this filter bank is

$$\tilde{P}(z) = \begin{bmatrix} \tilde{h}_e(z) & \tilde{h}_o(z) \\ \tilde{g}_e(z) & \tilde{g}_o(z) \end{bmatrix} = \begin{bmatrix} -\frac{1}{8}z^{-1} + \frac{3}{4} - \frac{1}{8}z & \frac{1}{4} + \frac{12}{4}z \\ -\frac{1}{2}z^{-1} - \frac{1}{2} & 1 \end{bmatrix}$$

Also based on conditions of perfect reconstructions of the complementary filters as described in Eq. (2), we can derive the corresponding synthesis filters as follows:

$$h(z) = -z^{-1}\tilde{g}(-z^{-1}) = \frac{1}{2}z^{-1} + 1 + \frac{1}{2}z,$$
$$g(z) = z^{-1}\tilde{h}(-z^{-1})$$
$$= -\frac{1}{8}z^{-3} - \frac{1}{4}z^{-2} + \frac{3}{4}z^{-1} - \frac{1}{4} - \frac{1}{8}z.$$

Thus $h = (\frac{1}{2}, 1, \frac{1}{2})$ and $g = (-\frac{1}{8}, -\frac{1}{4}, \frac{3}{4}, -\frac{1}{4}, -\frac{1}{8})$. Now based on the lifting scheme for factorization of the polyphase matrix, the possible factorization of $\tilde{P}(z)$ that leads to a band matrix multiplication is

$$\tilde{P}(z) = \begin{bmatrix} 1 & \frac{1}{4}(1+z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -\frac{1}{2}(1+z^{-1}) & 1 \end{bmatrix}$$

If the samples are numbered starting from 0, we can consider the even terms of the output stream as the samples of lowpass subband and the odd terms as the samples of highpass subband. Accordingly, we can interpret the above matrices in the time domain as $y_{2i+1} = b(x_{2i} + x_{2i+2}) + x_{2i+1}$ and $y_{2i} = a(y_{2i+1} + y_{2i+3}) + x_{2i}$, where $0 \leq i \leq N/2$ for an input stream x and output stream y both of length N, $a = -\frac{1}{2}$ and $b = \frac{1}{4}$. Note that the odd samples are calculated from even samples and even samples are calculated from the updated odd samples.

The other wavelet filter bank that has been proposed in JPEG2000 Part I standard is the (9,7) filter. The most efficient factorization of the polyphase matrix for (9,7) filter is as follows [4]:

$$\tilde{P}(z) = \begin{bmatrix} 1 & a(1+z^{-1}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ b(1+z) & 1 \end{bmatrix} \times \begin{bmatrix} 1 & c(1+z^{-1}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ d(1+z) & 1 \end{bmatrix} \begin{bmatrix} K & 0 \\ 0 & \frac{1}{K} \end{bmatrix}$$

where $a = -1.586134342$, $b = -0.05298011854$, $c = 0.8829110762$, $d = -0.4435068522$, $K = 1.149604398$.

In terms of banded matrix operation, the forward transform for (5,3) and (9,7) filters can be represented as $Y_{(5,3)} = XM_1M_2$ and $Y_{(9,7)} = XM_1M_2M_3M_4$ respectively, while the corresponding inverse transform are represented as $X = Y_{(5,3)}M_2M_1$ and $X = Y_{(9,7)}M_4M_3M_2M_1$, where

$$M_1 = \begin{bmatrix} 1 & a & 0 & . & . & . & . & . & . \\ 0 & 1 & 0 & 0 & . & . & . & . & . \\ 0 & a & 1 & a & 0 & . & . & . & . \\ . & 0 & 0 & 1 & 0 & 0 & . & . & . \\ . & . & 0 & a & 1 & a & 0 & . & . \\ . & . & . & 0 & 0 & 1 & 0 & 0 & . \\ . & . & . & . & 0 & a & 1 & a & 0 \\ . & . & . & . & . & 0 & 0 & 1 & 0 \\ 0 & . & . & . & . & . & 0 & a & 1 \end{bmatrix},$$

$$M_2 = \begin{bmatrix} 1 & 0 & 0 & . & . & . & . & . & . \\ 0 & 1 & b & 0 & . & . & . & . & . \\ 0 & 0 & 1 & 0 & 0 & . & . & . & . \\ . & 0 & b & 1 & b & 0 & . & . & . \\ . & . & 0 & 0 & 1 & 0 & 0 & . & . \\ . & . & . & 0 & b & 1 & b & 0 & . \\ . & . & . & . & 0 & 0 & 1 & 0 & 0 \\ . & . & . & . & . & 0 & b & 1 & 0 \\ 0 & . & . & . & . & . & 0 & 0 & 1 \end{bmatrix}$$

$$M_3 = \begin{bmatrix} 1 & c & 0 & . & . & . & . & . & . \\ 0 & 1 & 0 & 0 & . & . & . & . & . \\ 0 & c & 1 & c & 0 & . & . & . & . \\ . & 0 & 0 & 1 & 0 & 0 & . & . & . \\ . & . & 0 & c & 1 & c & 0 & . & . \\ . & . & . & 0 & 0 & 1 & 0 & 0 & . \\ . & . & . & . & 0 & c & 1 & c & 0 \\ . & . & . & . & . & 0 & 0 & 1 & 0 \\ 0 & . & . & . & . & . & 0 & c & 1 \end{bmatrix},$$

$$M_4 = \begin{bmatrix} 1 & 0 & 0 & . & . & . & . & . & . \\ 0 & 1 & d & 0 & . & . & . & . & . \\ 0 & 0 & 1 & 0 & 0 & . & . & . & . \\ . & 0 & d & 1 & d & 0 & . & . & . \\ . & . & 0 & 0 & 0 & 1 & 0 & . & . \\ . & . & . & 0 & d & 1 & d & 0 & . \\ . & . & . & . & 0 & 0 & 1 & 0 & 0 \\ . & . & . & . & . & 0 & d & 1 & 0 \\ 0 & . & . & . & . & . & 0 & 0 & 1 \end{bmatrix}$$

In fact, most of the popular wavelet filters are decomposed either into 2 or 4 matrices (primal and dual). For example, the wavelet filters C(13, 7), S(13, 7), (2, 6), (2, 10) can be decomposed into 2 matrices and (6, 10) can be decomposed in 4 matrices as have been described in detail in [12].

The lifting-based DWT has many advantages over the convolution based approach. Some of them are as follows.

- Lifting-based DWT typically requires less computation (up to 50%) compared to the convolution based approach. However the savings depends upon the length of the filters.
- During the lifting implementation, no extra memory buffer is required because of the in-place computation feature of lifting. This is particularly suitable for hardware implementation with limited on-chip memory.
- The lifting based approach offers integer to integer transformation suitable for lossless image compression.
- In lossless transformation mode, the boundary extension of the input data can be avoided because the original input can be exactly reconstructed by integer to integer lifting transformation.

3. Lifting Architectures for 1D DWT

The data dependencies in the lifting scheme can be explained with the help of an example of DWT filtering with four factors (or four lifting steps). The four lifting steps correspond to four stages as shown in Fig. 5. The intermediate results generated in the first two stages for the first two lifting steps are subsequently processed to produce the high-pass (HP) outputs in the third stage, followed by the low-pass (LP) outputs in the fourth stage. (9,7) filter is an example of a filter that requires four lifting steps. For the DWT filters requiring only two factors, such as the (5,3) filter, the intermediate two stages can simply be bypassed.

3.1. Direct Mapped Architecture [6]

A direct mapping of the data dependency diagram into a pipelined architecture was proposed by Liu et al. in [6] and described in Fig. 6. The architecture is designed with 8 adders (A1–A8), 4 multipliers (M1–M4), 6 delay elements (D) and 8 pipeline registers (R). There

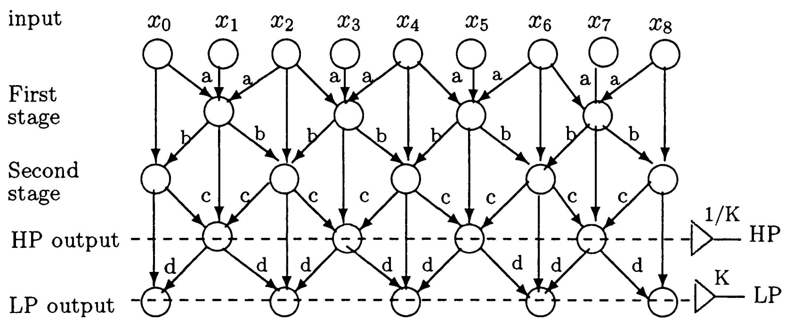


Figure 5. Data dependency diagram for lifting of filters with four factors.

are two input lines to the architecture: one that inputs even samples $\{x_{2i}\}$), and the other one that inputs odd samples $\{x_{2i+1}\}$. There are four pipeline stages in the architecture. In the first pipeline stage, adder A1 computes $x_{2i} + x_{2i-2}$ and adder A2 computes $a(x_{2i} + x_{2i-2}) + x_{2i-1}$. The output of A2 corresponds to the intermediate results generated in the first stage of Fig. 5. The output of adder A4 in the second pipeline stage corresponds to the intermediate results generated in the second stage of Fig. 5. Continuing in this fashion, adder A6 in the third pipeline stage produces the high-pass output samples, and adder A8 in the fourth pipeline stage produces the low-pass output samples. For lifting schemes that require only 2 lifting steps, such as the (5,3) filter, the last two pipeline stages need to be bypassed causing the hardware utilization to be only 50% or less. Also, for a single read port memory, the odd and even samples are read serially in alternate clock cycles and buffered. This slows down the overall pipelined architecture by 50% as well.

A similar pipelined architecture for the (9,7) wavelet has been proposed by Jou et al. in [7].

3.2. Folded Architecture [8]

The pipelined architecture in Fig. 6 can be further improved by carefully folding the last two pipeline stages into the first two stages as shown in Fig. 7. The architecture proposed by Lian, et al. in [8)] consists of two pipeline stages, with three pipeline registers, R1, R2 and R3. In the (9,7) type filtering operation, intermediate data (R3) generated after the first two lifting steps (phase 1) are folded back to R1 (as shown in Fig. 7) for computation of the last two lifting steps (phase 2). The architecture can be reconfigured so that computation of the two phases can be interleaved by selection of appropriate data by the multiplexors. As a result, two delay registers (D) are needed in each lifting step in order to properly schedule the data in each phase. Based on the phase of interleaved computation, the coefficient for multiplier M1 is either a or c, and similarly the coefficient for multiplier M2 is b or d. The hardware utilization of this architecture is always 100%. Note that for the (5,3) type filter operation, folding is not required.

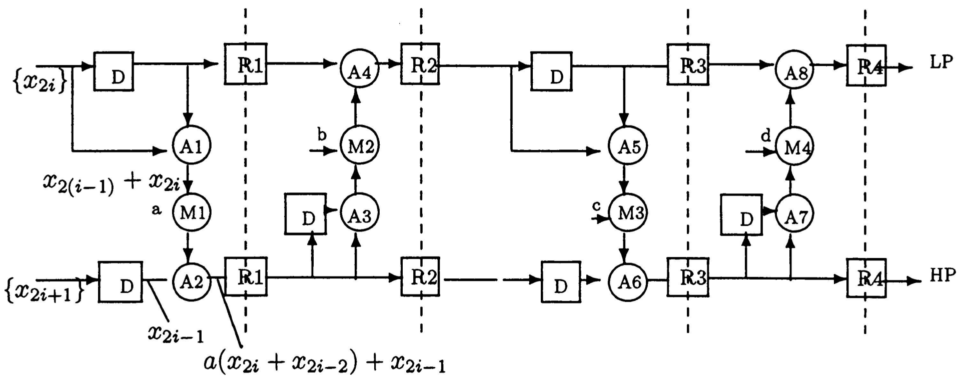


Figure 6. The direct mapped architecture in [6].

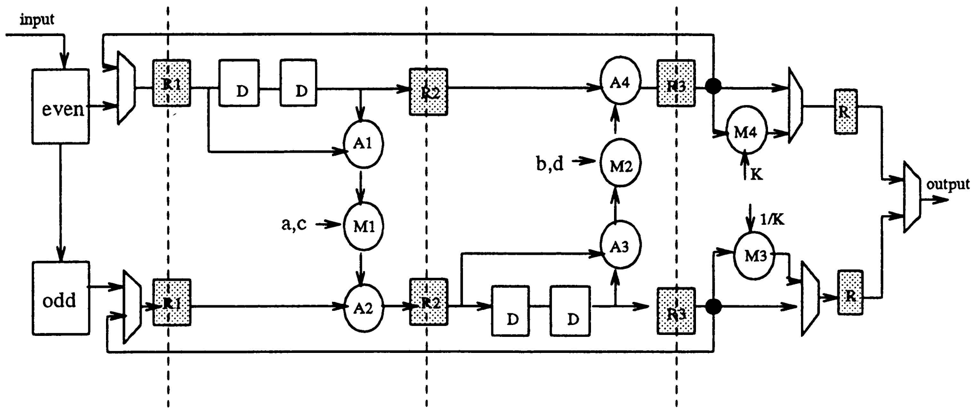


Figure 7. The folded architecture in [8].

3.3. MAC Based Programmable Architecture [10]

A programmable architecture that implements the data dependencies represented in Fig. 5 using four MACs (Multiply and Accumulate) and nine registers has been proposed by Chang et al. in [10]. The algorithm is executed in two phases as shown in Fig. 8. The data-flow of the proposed architecture can be explained in terms of the register allocation of the nodes. The computation and allocation of the registers in phase 1 are done in the following order

$$R0 \leftarrow x_{2i-1}; \quad R2 \leftarrow x_{2i};$$
$$R3 \leftarrow R0 + a(R1 + R2);$$
$$R4 \leftarrow R1 + b(R5 + R3);$$
$$R8 \leftarrow R5 + c(R6 + R4);$$
$$Output_{LP} \leftarrow R6 + d(R7 + R8); \quad Output_{HP} \leftarrow R8$$

Similarly, the computation and register allocation in phase 2 are done in the following order

$$R0 \leftarrow x_{2i+1}; \quad R1 \leftarrow x_{2i+2};$$
$$R5 \leftarrow R0 + a(R2 + R1);$$
$$R6 \leftarrow R2 + b(R3 + R5);$$
$$R7 \leftarrow R3 + c(R4 + R6);$$
$$Output_{LP} \leftarrow R4 + d(R8 + R7); \quad Output_{HP} \leftarrow R7$$

As a result, two samples are input per phase and two samples (LP and HP) are output at the end of every phase. For 2D DWT implementation, the output samples are also stored into a temporary buffer for filtering in the vertical dimension.

3.4. Flipping Architecture [11]

While conventional lifting-based architectures require fewer arithmetic operations, they sometimes have long critical paths. For instance, the critical path of the lifting-based architecture for the (9,7) filter is $4T_m + 8T_a$ while that of the convolution implementation is $T_m + 4T_a$. One way of improving this is by pipelining which results in a significant increase in the number of registers. For instance, to pipeline the lifting-based

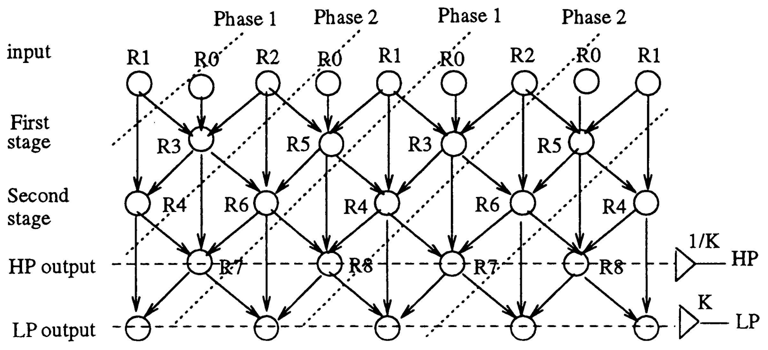


Figure 8. Data-flow and register allocation of the MAC based architecture in [10].

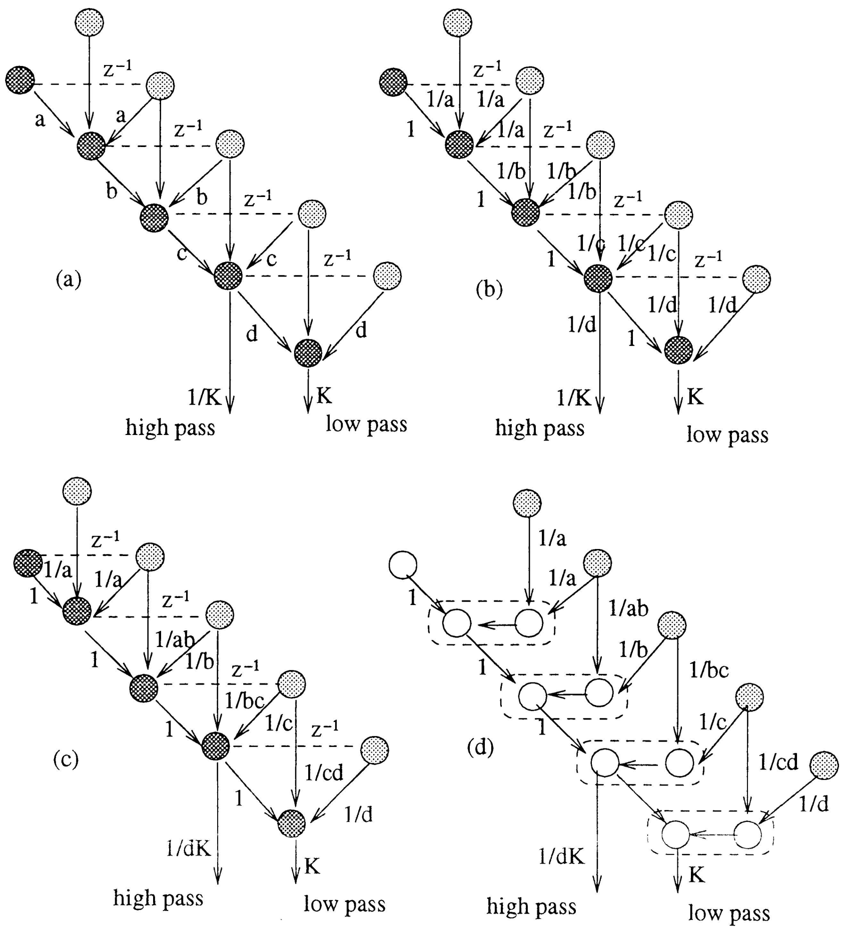


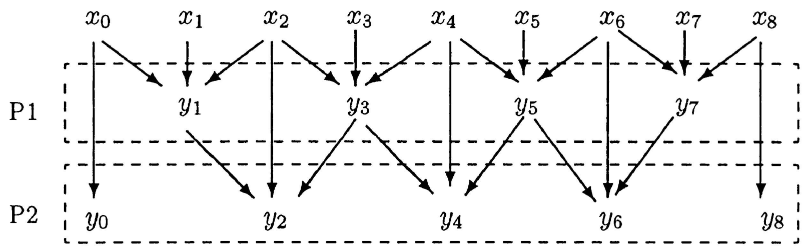
Figure 9. A flipping architecture [11]. (a) Original architecture, (b)–(c) Scaling the coefficients to reduce the number of multiplications, (d) Splitting the three-input addition nodes to two-input nodes.

(9,7) filter such that the critical path is $T_m + 2T_a$, 6 additional registers are required.

Recently, Huang et al. [11] proposed a very efficient way of solving the timing accumulation problem. The basic idea is to remove the multiplications along the critical path by scaling the remaining paths by the inverse of the multiplier coefficients. Fig. 9(a)–(c) describes how scaling at each level can reduce the multiplications in the critical path. Figure 9(d) further splits the three input addition nodes into two 2-input adders. The critical path is now $T_m + 5T_a$. The minimum critical path of T_m can be achieved by 5 pipelining stages using 11 pipelining registers (not shown in the figure). Detailed hardware analysis of lossy (9,7), integer (9,7) and (6,10) filters have been included in [11]. Further more, since the flipping transformation changes the round-off noise considerably, techniques to address precision and noise problems have also been addressed in [11].

3.5. *Generalized Architecture [12]*

The architecture proposed by Andra et al. [12] is an example of a highly programmable architecture that can support a large set of filters. These include filters (5,3), (9,7), C(13,7), S(13,7), (2,6), (2,10), and (6,10). Since the data dependencies in the filter computations can be represented by at most four stages, the architecture consists of four processors, where each processor is assigned computations of one stage. Figure 10(a) describes the assignment of computation to two processors, P1 and P2, for the (5,3) filter which can be represented by two stages.


P1
P2
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8
y_1 y_3 y_5 y_7
y_0 y_2 y_4 y_6 y_8

(a) Processor assignment for the (5,3) filter

Cycle	Processor 1 (P1)			Processor 2 (P2)		
	Adder1	Shifter	Adder2	Adder1	Shifter	Adder2
1	–	–	–	–	–	–
2	$x_0 + x_2$	–	–	–	–	–
3	$x_2 + x_4$	R1	–	–	–	–
4	$x_4 + x_6$	R1	Rs-x_1=y_1	–	–	–
5	$x_6 + x_8$	R1	Rs-x_3=y_3	–	–	–
6		R1	Rs-x_5=y_5	y_1, y_3	–	–
7		–	Rs-x_7=y_7	y_3, y_5	R1	y_0
8				y_5, y_7	R1	Rs+x_2
9					R1	Rs+x_4
10						Rs+x_6

(b) Partial schedule for the (5,3) filter implementation

Figure 10. Processor assignment and partial schedule for the (5, 3) filter implementation in the Generalized architecture in [12].

The processor architecture consists of adders, multipliers and shifters that are interconnected in a manner that would support the computational structure of the specific filter. Figure 11 describes the processor architectures for the (5,3) filter and the (9,7) filter. While the (5,3) filter architecture consists of two adders, and a shifter, the (9,7) filter architecture consists of two adders and a multiplier. Figure 10(b) describes part of the schedule for the (5,3) filter. The schedules are generated by mapping the data dependency graph onto the

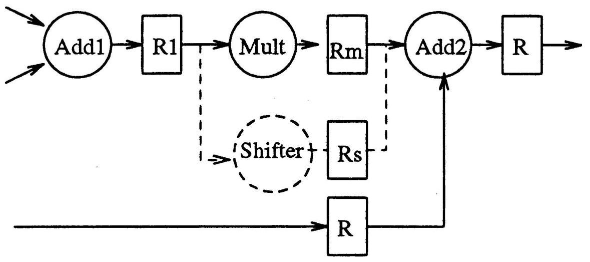


Figure 11. Processor architecture for the (5,3) and (9,7) filters in [12].

resource-constrained architecture. It is assumed that the delays of each adder, shifter and the multiplier are 1, 1 and 4 time units respectively. For example, Adder1 of P1 adds the elements (x_0, x_2) in the 2nd cycle and stores the sum in register $R1$. The shifter reads this sum in the next cycle (3rd cycle), carries out the required number of shifts (one right shift as $a = -0.5$) and stores the data in register Rs. The second adder (Adder2) reads the value in Rs and subtracts the element x_1 to generate y_1 in the next cycle. To process $N = 9$ data, the P1 processor takes four cycles. Adder 1 in P2 processor starts computation in the sixth cycle. The gaps in the schedules for P1 and P2 are required to store the zeroth element of each row.

3.6. *Recursive Architecture [14]*

Most of the traditional DWT architectures compute the ith level of decomposition upon completion of the $(i - 1)$th level of decomposition. However in

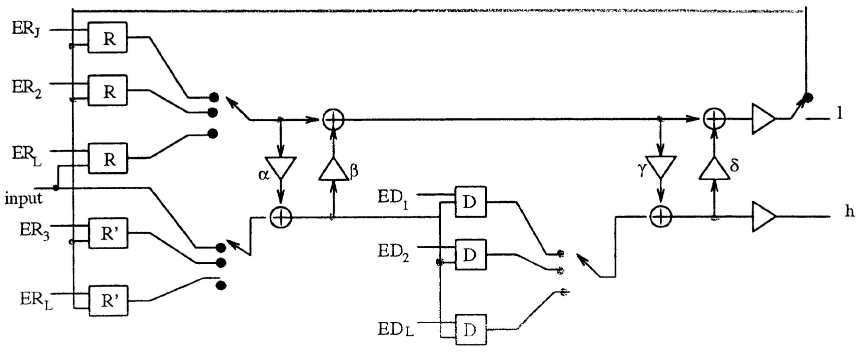


Figure 12. The recursive architecture in [14].

multiresolution DWT, the number of samples to be processed in each level is always half of the size in the previous level. Thus it is possible to process multiple levels of decomposition simultaneously. This is the basic principle of a *recursive architecture* that was first proposed for a convolution based DWT in [13] and applied for lifting based DWT in [14, 15]. Here computations in higher levels of decomposition is initiated as soon as enough intermediate data in low-frequency subband is available for computation. The proposed architecture for a 3-level decomposition of an input signal using Daubechies-4 DWT is shown in Fig. 12.

The basic circuit elements used in this architecture are delay elements, multipliers and MAC units which are in turn designed using a multiplier, an adder and two shifters. The multiplexors M1 and M2 select the even and odd samples of the input data as needed by the lifting scheme. S1, S2 and S3 are the control signals for data flow of the architecture. The select signal (S1) of the multiplexors is set to 0 for the first level of computation and is set to 1 during the second or third level computation. The switches S2 and S2 select the input data for the second and third level of computation. The multiplexor M3 selects the delayed samples for each level of decomposition based on the clocked signals shown in Fig. 12.

A recursive architecture for 1D implementation of the (5,3) filter has been proposed in [9]. The architecture has hardware complexity identical to [15] but is claimed to be more regular. The topology is similar to a scan chain, and thus can be easily modified to support testable scan-based designs.

3.7. Dual Scan Architecture in [15]

In [15], Liao et al. presented a 1D *dual scan architecture* (DSA) for DWT that achieves 100% datapath hardware utilization (for special cases) by processing two independent data streams together using shared functional blocks in an interleaved fashion.

The architecture consists of a processing element that implements the conventional lifting scheme, one memory unit and input and output switches as shown in Fig. 13. The input switches are connected to the input signals of the processing element when it computes the first stage of lifting and are connected to the memory unit when the processing element performs the other stages of lifting (the stages are shown in the data dependency diagram in Fig. 5). The switch SW0 separates the low-frequency coefficients of the two input signals. The switch SW1 is connected to the output only after completion of the final stage of lifting. During the DWT computation, the input samples are shifted in and the low-frequency coefficients are stored in the internal memory. After all the input samples in one stage are processed, the stored coefficients are retrieved to start computation in the next stage. Since DSA performs useful calculation in every clock cycle, its hardware utilization for the processing element is effectively 100%.

3.8. DSP Type Architecture [17]

A filter independent DSP type parallel architecture has been proposed by Martina et al. in [17]. The

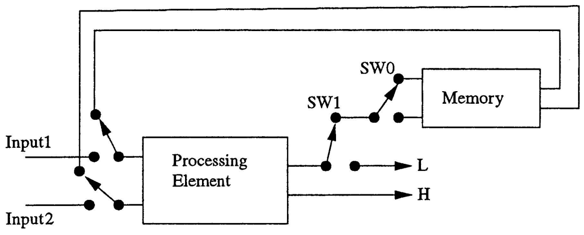


Figure 13 The dual scan architecture in [15].

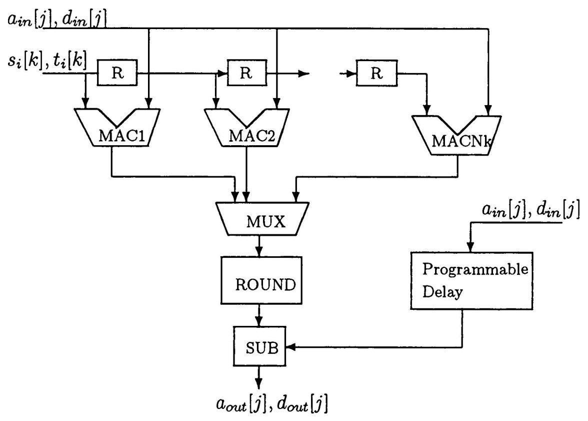


Figure 14. Parallel MAC architecture for lifting [17].

architecture consists of $N_t = \max_i\{k_{s_i}, k_{t_i}\}$ number of MAC (Multiply-Accumulate) units, where k_{s_i} and k_{t_i} are length of the primal and dual lifting filters s_i and t_i respectively, in step i of the lifting factorization. The architecture is shown in Fig. 14. The architecture essentially computes the following two streams in each lifting step.

$$a_{\text{out}}[j] = a_{\text{in}}[j] - \left\lfloor \sum_k d_{\text{in}}[j-k] \cdot s_i[k] + \frac{1}{2} \right\rfloor,$$

$$d_{\text{out}}[j] = d_{\text{in}}[j] - \left\lfloor \sum_k a_{\text{out}}[j-k] \cdot t_i[k] + \frac{1}{2} \right\rfloor,$$

where a_{in} and d_{in} are two input sub-streams formed by the even and odd samples of the original input signal stream x. It is obvious that streams a_{in} and b_{in} are not processed together in this architecture; while one is processed the other has to be delayed enough to guarantee a consistent subtraction at the end of the lifting step. The above architecture is designed to compute n_t simultaneous partial convolution products selected by the MUX, where n_t is the length of filter tap for the lifting step being currently executed in the architecture. After n_t clock cycles, the first filtered sample is available for rounding operation at the output of the first MAC_1 and subsequent samples are obtained in consecutive clock cycles from the subsequent MAC units ($MAC_2, \ldots, MAC_{nt}$). The 'programmable delay' is a buffer that guarantees the subtraction consistence to execute corresponding $a_{\text{out}}[j]$ and $d_{\text{out}}[j]$ samples at the output. The ROUND unit in Figure 14 computes the floor function shown in the lifting equations and the SUB unit processes the corresponding subtraction operations. The architecture can be programmed to support a wide range of filters including (9,7), (10,18), (13,11), (6,10), (5,3) and (9,3).

3.9. *Comparison of Performance of the 1D Architectures*

A summary of the hardware and timing requirements of the different (9,7) filter implementations for data size N is presented in Fig. 15. The hardware complexity has been compared with respect to the data path. The memory size and organization required to support multiple levels of decomposition has not been listed in most of the architectures, and hence not included here. An estimate of the controller complexity has also been included. The timing performance has been compared with respect to two parameters: the number of clock cycles to compute L levels of decomposition and the clock period (i.e., the delay in the critical path). The notation T_m stands for the delay of a multiplier, T_a the delay of an adder, etc.

In terms of hardware complexity, the folded architecture in [8] is the simplest and the DSP-based architecture in [17] is the most complex. All other architectures have comparable hardware complexity and primarily differ in the number of registers and multiplexor circuitry. The control complexity of the architecture in [6] is very simple. In contrast, the number of switches, multiplexors and control signals used in the architectures of [15, 17] is quite large. The control complexity of the remaining architectures is moderate.

In terms of timing performance, the architectures in [6, 8, 10–12] are all pipelined, with the architectures in [11, 12] having the highest throughput ($1/T_m$). The architecture in [14] has fewer cycles since it is RPA based, but its clock period is higher. The architecture in [11] has the lowest computation delay though it may not be apparent from Fig. 15.

Finally, all the architectures with the exception of [14] compute all the outputs of one level before starting computations of the next level. The architecture in [14] is the only one that adopts an RPA based approach and intersperses the computations of the higher levels with those of the first level. So it is likely that the memory requirements of [14] would be lower than the others.

4. Two Dimensional DWT Architecture

Generally, 2D wavelet filters are separable functions. A straight-forward approach for 2D implementation

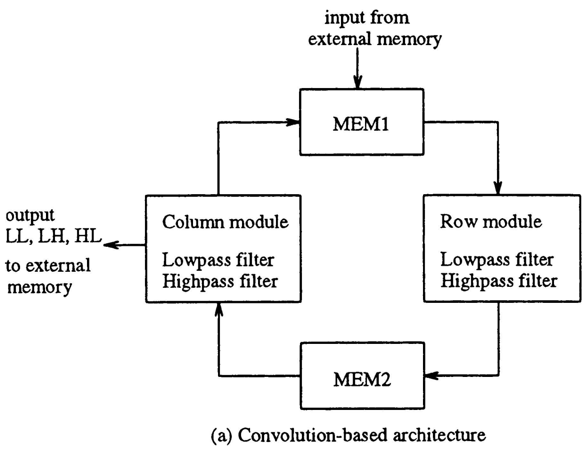
Architecture	Datapath	Timing		Control
		Number of cycles	Clock period	
Direct mapped[6]	4 mult, 2 scaling mult, 8 adders, 8 registers, 6 delay units	$4 + 2N(1 - 1/2^L)$	$T_m + 2T_a$	Simple
Folded [8]	4 mult, 2 scaling mult, 8 adders, 8 registers, 6 delay units	$4 + 2N(1 - 1/2^L)$	$2T_m + 2T_a$	Moderate
MAC [10]	4 multiply-accumulate, 2 scaling mult, 9 registers	$4 + 2N(1 - 1/2^L)$	$T_m + 2T_a$	Moderate
Flipping [11]	4 mult, 2 scaling mult, 8 adders, 10 registers, 6 shifters	$5 + 2N(1 - 1/2^L)$	T_m	Moderate
Generalized [12]	4 processors (each with 1 mult, 2 adders, 2 registers), 2 scaling mult	$12 + 2N(1 - 1/2^L)$	T_m	Moderate
Recursive [14]	4 mult, 2 scaling mult, 4 adders, 7 registers, 3 delay units, 6 mux	$N + L + 2^L$	$4T_m + 8T_a$	Complex
DSA [15]	2 processors (each with 4 mult, 2 scaling mult, 4 adders, 7 reg, 3 delay units)	$N + L$	$4T_m + 8T_a$	Moderate
DSP [17]	2 MACs(each with 2 mul, 2 adders, 12 reg, output buffer), ROUND, SUB units, 2 scaling mult, prog. delay	$N(1 - 1/2^L)$, where $k = T_m + 2T_a + T_{rd} + T_{sub} + T_{buffer}$	$2k$	Complex

Figure 15. Hardware and timing comparison of the 1D DWT architectures for the (9, 7) filter computation on an input size *N* with *L* levels of decomposition.

is to first apply the 1D DWT row-wise (to produce L and H subbands) and then column-wise to produce four subbands LL, LH, HL and HH as shown in Fig. 3 in each level of decomposition. Obviously, the processor utilization is a concern in direct implementation of this approach because it requires all the rows be filtered before the columnwise filtering can begin and thus it requires a size of memory buffer of the order of the image size. The alternative approach is to begin the column-processing as soon as sufficient number of rows have been filtered. The column-wise processing is now performed on these available lines to produce wavelet coefficients row-wise. The overview of the two-dimensional architecture for convolution based DWT is shown in Fig. 16(a). The row module reads the data from MEM1, performs DWT along the rows and writes the data into MEM2. The column module reads the data from MEM2, performs DWT along the columns and writes 'LL' data to MEM1 and 'LH', 'HL', 'HH' data to external memory.

A similar approach can be implemented for the lifting scheme as well. The basic idea of lifting based approach for DWT implementation is to replace the parallel low-pass and high-pass filtering of traditional approach by a sequence of alternating smaller filters. The computations in each filter can be partitioned into prediction (dual lifting) and update (primal lifting) stages as shown in Fig. 16(b). Here the row module reads the data from MEM1, performs the DWT along the rows ('H' and 'L') and writes the data into MEM2. The prediction filter of the column module reads the data from MEM2, performs column-wise DWT along alternate rows ('HH' and 'LH') and writes the data into MEM2 in [12] (and into MEM1 in [21]); the update filter of the column module reads the data from MEM2 in [12] (and MEM1 in [21]), performs column-wise DWT along the remaining rows, and writes the 'LL' data into MEM1 for higher octave computations and 'HL' data to external memory. Note that this is a generic architectural flow and is the backbone of the existing 2D architectures.

An important consideration in the design of 2D architectures is the memory configuration. A trade-off exists between the size of the internal memory and the frame memory access bandwidth. The size of the internal memory is again a function of the way the frame memory is scanned. In Section 4.1, we describe the existing scanning techniques along the lines of [23, 22].



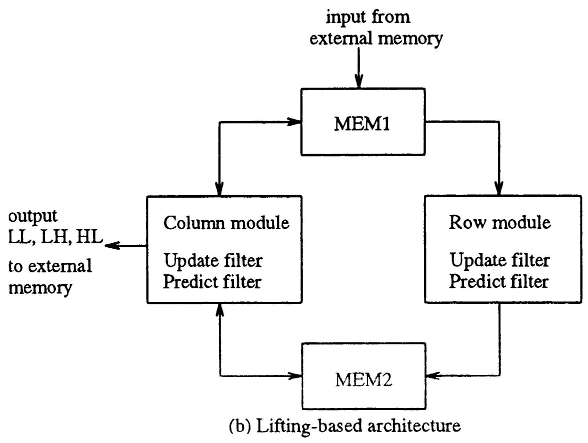


Figure 16. Overview of convolution and lifting-based 2D DWT architecture.

Then we describe three representative 2D DWT architectures, namely, the dedicated architecture for the (4,2) filter [21], the generalized architecture [12] and the recursive architecture [15], and compare them with respect to hardware and timing complexities.

4.1. *Memory Scan Techniques*

The memory scan techniques can be broadly classified into line-based scan, block-based scan and stripe-based scan. Though most of the existing architectures are based on line scan, we describe all three techniques to (possibly) facilitate development of new 2D DWT architectures.

4.1.1. Line-based Scan. In line-based scan, the scan order is raster scan. An internal line buffer of size LN is required, where N is the number of pixels in a row

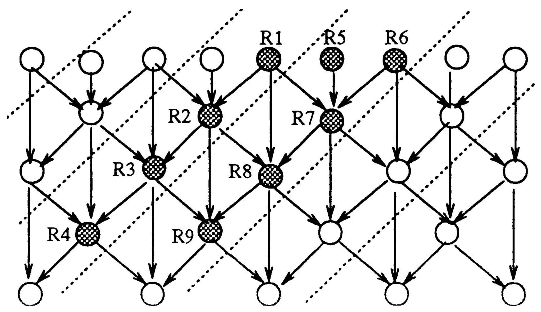


Figure 17. Line-based scan method.

and L is the number of rows that are required for that particular filter. A line-based implementation of the traditional convolution based wavelet transform has been discussed in great detail in [18]. For lifting based architectures, the value of L can be determined as in [22] by considering the data dependency graph (see Fig. 17). This is an extension of the 1D data dependency graph (see Fig. 5) with a node now corresponding to a row of pixels. Note that several rows of data corresponding to R2-R4 and coefficients corresponding to R1 and R5 have to be stored. When a new row of data corresponding to R6 is available, another column operation can be initiated. After this column operation, data in R7–R9 are stored for the next column operation. According to the implementation in [22], the line buffer needs to store six rows of data. The implementation in [23] as well as that in [20] requires only four rows of data to be stored. A detailed analysis of the memory requirements for line scan implementations of both forward and inverse transforms are presented in [19].

4.1.2. Block-based Scan. In block-based scan, the frame memory is scanned block-by-block and the DWT coefficients are also computed block-by-block. Figure 18 shows two configurations of block based methods where the blocks are scanned in the row direction first. In the non-overlapped configuration, the

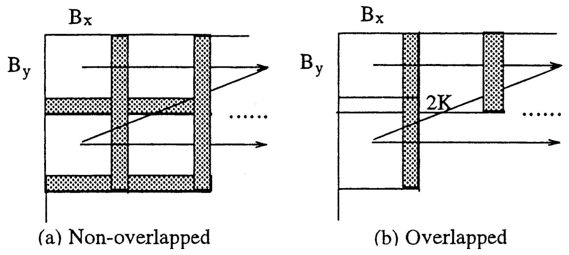


Figure 18. Block-based scan method [23].

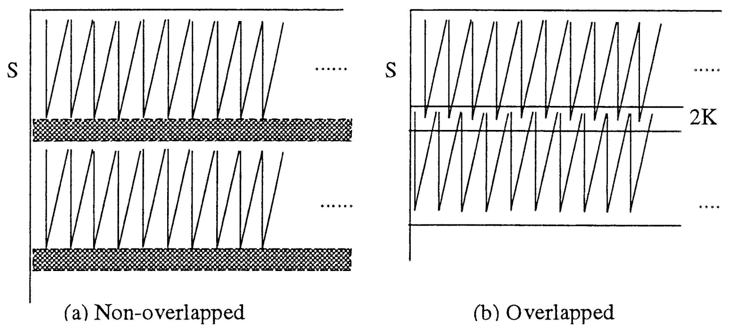


Figure 19. Stripe-based scan method [23].

blocks are not overlapped with each other and in the overlapped configuration, the blocks are overlapped by 2K pixels in the column direction. Here $K = \lfloor (L-1)/2 \rfloor$, where L is the number of DWT filter taps. In both cases, intermediate data have to be stored between two adjacent blocks as shown in grey in Fig. 18. The size of the internal buffer for one level for the non-overlapped case is $LN + LB_y$. The first term, LN, is due to the column-wise intermediate data and the second term, LB_y is due to the intermediate data between adjacent blocks in a row. The size of the internal buffer can be reduced to only LB_y if the column-wise intermediate data is not stored and instead the data is read from the frame memory as needed. The size of the internal buffer for the overlapped case can also be reduced to LB_y at the expense of increasing the number of frame memory reads to $N^2B_y/(B_y-2K)$ [23]. However, this scheme is not directly applicable to multi-level architectures.

The block-based technique proposed in [20] first performs filtering operation on neighboring data blocks independently and later combines the partial boundary results together. Two boundary post-processing techniques are proposed - overlap-state sequential which reduces the buffer size for sequential processing and split-and-merge which reduces the interprocessor delay in parallel implementations.

4.1.3. Stripe-based Scan. The stripe-based scan is equivalent to the line-based scan with $B_x = N$. In other words, the stripe is a very wide block with width N and height S. As in the case of block-based scan, there are two categories, namely, the non-overlapped stripe-based scan also referred to as the optimal Z-scan in [22] and shown in Fig. 19(a), and the overlapped stripe-based scan shown in Fig. 19(b). The non-overlapped stripe-based scan has an internal buffer of size $LN + LS$ and N^2 frame memory READ accesses. In contrast, the overlapped stripe-based scan has a significantly smaller internal buffer of size LS and $N^2S/(S-2K)$ frame memory READ accesses.

4.2. *(4,2) Filter Architecture in [21]*

A dedicated architecture for 2D DWT using the (4,2) filter from the Deslauriers-Dubuc family has been proposed by Ferretti and Rizzo in [21]. The architecture, shown in Fig. 20. It consists of two parallel filters to

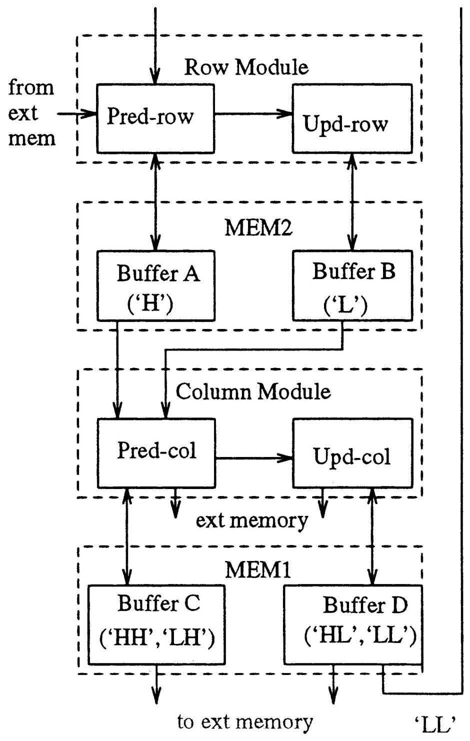


Figure 20. Block diagram of the (4,2) filter architecture in [21].

compute the predict and update values along the rows (*Pred-row, Upd-row*), two parallel filters to compute the predict and update values along the columns (*Pred-col, Upd-col*), and four buffers A, B, C, D, to hold the intermediate data to support the pipelined computations. The buffers are dual-ported and are organized such that 4 words can be accessed simultaneously. Each filter consists of multipliers ($L_g = 4$ for predict filters and $L_h = 2$ for update filters), adders, shifters and internal buffers (proportional to l_g and L_h) to streamline the computations.

Pred-row computes on $L_g = 4$ data, $L_g - 1$ of which are stored in its internal buffer. It computes the 'H' values. The *Upd-row* requires $L_h = 2$ 'H' values to compute a 'L' value. It obtains these by reading the last value produced by *Pred-row* and storing the other $L_h - 1$ in internal registers. It picks up the primary input value from the internal buffer in *Pred-row*.

Pred-col performs the same basic operations as *Pred-row*, though working on columns. It reads L_g even position 'H' values along the columns. It produces a new row of wavelet coefficients for every two rows produces by *Pred-row*. During the time *Pred-row* produces 'H' values for odd-indexed rows, *Pred-col* computes on the 'L' values generated by *Upd-row*.

The architecture utilization is only 50% if we only consider the computations in the first level. The higher level computations can thus be easily interspersed with the first level computations using a RPA-based approach. In fact, once the unit delay for any level is determined, the schedule can be easily obtained. Please refer to [21] for details of the schedules, memory sizes, number of read/write accesses, etc.

4.3. *Generalized 2D DWT Architecture in [12]*

The architecture proposed by Andra et al. [12] is more generalized and can compute a large set of filters for both the 2D forward and inverse transforms. It supports two classes of architectures based on whether lifting is implemented by one or two lifting steps. The M2 architecture corresponds to implementation using one lifting step or two factorization matrices, and the M4 architecture corresponds to implementation by two lifting steps or four factorization matrices.

The dataflow of the M2 architecture that is used to implement the wavelet filters (5,3), C(13,7), S(13,7), (2,6), (2,10) is similar to that in Fig. 16(b). A block diagram of the M2 architecture is shown in Fig. 21.

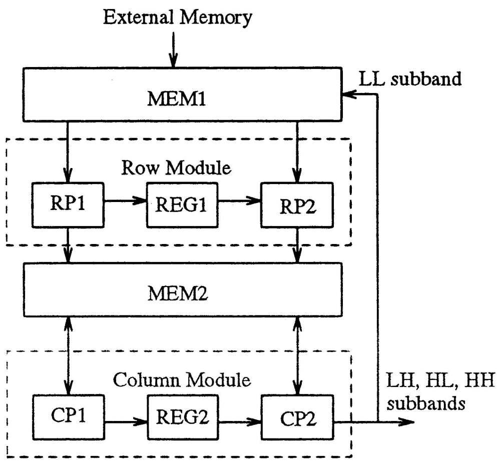


Figure 21. Block diagram of the generalized M2 architecture in [12].

It consists of the row and column computation modules and two memory units, MEM1 and MEM2. The row module consists of two processors RP1 and RP2 along with a register file REG1, and the column module consists of two processors CP1 and CP2 along with a register file REG2. All the four processors RP1, RP2, CP1, CP2 in the proposed architecture consists of 2 adders, 1 multiplier and 1 shifter as shown in Fig. 11.

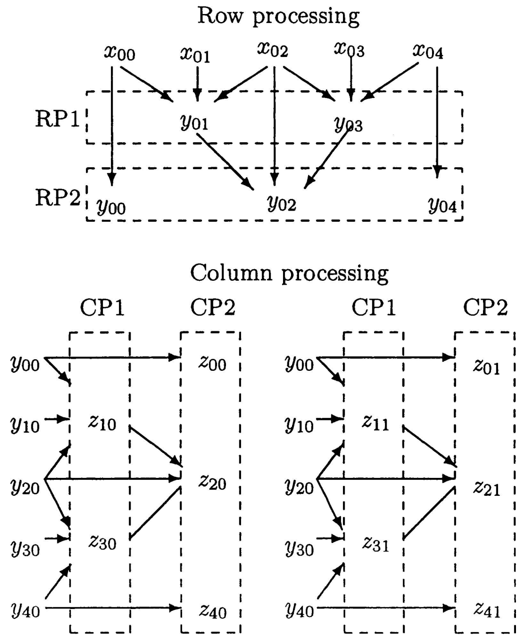


Figure 22. Data access patterns for the row and column modules for the (5,3) filter with $N = 5$ in the 2D DWT architecture in [12].

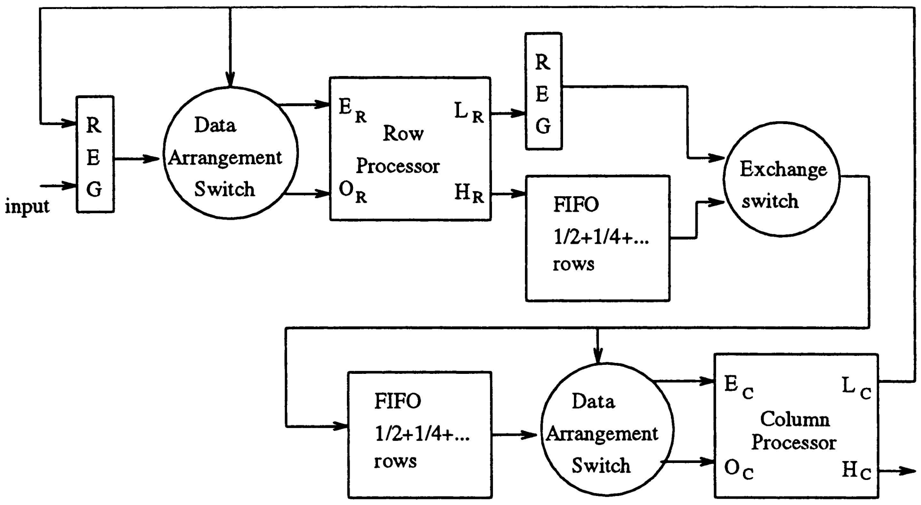


Figure 23. Block diagram of the 2D recursive architecture in [15].

For the M2 architecture, RP1 and CP1 are predict filters and RP2 and CP2 are update filters.

Figure 22 illustrates the data access pattern for the (5,3) filter with $N = 5$. RP1 calculates the *high-pass* (odd) elements along the rows, $y_{01}, y_{03}, \ldots$, while RP2 calculates the *low-pass* (even) elements along the rows, $y_{00}, y_{02}, y_{04}, \ldots$, CP1 calculates the *high-pass* and *low-pass* elements $z_{10}, z_{11}, \ldots; z_{30}, z_{31}, \ldots$ along odd rows and CP2 calculates *high-pass and low-pass* elements $z_{00}, z_{01}, \ldots; z_{20}, z_{21}, \ldots; z_{40}, z_{41}, \ldots$ along the even rows. Note that CP1 and CP2 start computations as soon as the required elements are generated by RP1 and RP2.

The memory modules, MEM1 and MEM2, are both dual port with one read and one write port, and support two simultaneous accesses per cycle. MEM1 consists of two banks and MEM2 consists of four banks. The multi-bank structure increases the memory bandwidth and helps support highly pipelined operation. Details of the memory organization and size, register file, and schedule for the overall architecture with specific details for each constituent filter have been included in [12].

The dataflow of the M4 architecture that is used to implement the filters (9,7), (6,10) is quite different. Since this is a generalized architecture with the hardware in the row and column modules fixed, the computations span two passes. In the first pass, the row-wise computations are performed using both the modules. Module 1 reads the data from MEM1, executes the first two matrix multiplications, and writes the result into MEM2. Module 2 executes the next two matrix multiplications and writes the result into MEM1. In the second pass, the transform is computed along the columns. Once again, Module 1 executes the first two matrix multiplications and Module 2 executes the next two matrix multiplications.

4.4. *2D Recursive Architecture in [15]*

The 2D recursive architecture proposed by Liao et al. [15] is built on top of the 1D recursive architecture proposed by the same authors in [14, 15] and presented in Section 3.6. As in the 1D case, the computation of all the lifting stages are interleaved to increase the hardware utilization. A simplified block diagram of this architecture is shown in Fig. 23. The column-processor and the row-processor are similar to the 1D recursive architecture processor. The image is input to the row-processor in raster scan format. When the row-processor processes the even rows, the high- and low-frequency DWT coefficients of the odd rows are shifted into their corresponding *first-in first-out* FIFO registers. The use of two FIFOs to separately store high frequency and low frequency components results in lower controller complexity. When the row processor processes the odd lines, the low-frequency DWT coefficients of the current line and lines previously stored in the FIFOs are sent to the column processors. The column-processor starts calculating the vertical DWT in zigzag scan format after one row delay. The

Architecture	Hardware		Timing		Control
	Datapath	Int. memory	Number of cycles	Clock Period	
(4,2) filter [21]	10 mult, 8 adders, 5 inc, 4 shifters, 47 data reg, 8 control reg, 6 mux	25N+7(L-2)	not provided	$T_m + 2T_a + T_s$	Moderate
Generalized [12] (9,7)	4 mult, 2 scaling mult, 8 adders, 4 shifters, 16 reg	$N^2 + 34$	$\approx 4/3N^2 + 2L$	T_m	Moderate
Recursive [15] (9,7)	8 mult, 4 scaling mult, 8 adders, 14 reg, large no of multiplexors	$4N$	$\approx N^2 + N + 2L$	$4T_m + 8T_a$	Complex
Folded RPA[16] (9,7)	9 mul, 2 adders, multiplexors, registers	$12N$	similar to [14]	$4T_m + 4T_a$	Complex

Figure 24. Hardware and timing comparison of the 2D DWT Architectures on an input of size $N \times N$ with L levels of decomposition.

computations are arranged in a way that the DWT coefficients of the first stage are interleaved with the other stages. The arrangement is done with the help of the data arrangement switches at the input to the row and column processors, and the exchange switch.

A mix of the principles of *recursive pyramid algorithm* (RPA) [13] and *folded architecture* has been adopted by Jung et al. to design a 2D architecture for lifting based DWT in [16]. The row-processor is a 1D folded architecture and does row-wise computations in the usual fashion. The column processor is responsible for filtering along the columns at the first level and filtering along both the rows and the columns at the higher levels. It does this by employing RPA scheduling and achieves very high utilization. The utilization of the row processor is 100%, and that of the column processor is 83% for 5-level decomposition.

4.5. *Other 2D DWT Architectures*

Several of the 1D DWT architectures proposed in Section 3 can be extended to compute 2D DWT using the row-column method. If a frame memory based approach is used where the original samples are replaced by the calculated coefficients, no additional memory is required. However, an address generator is needed to provide the proper access addresses in order to read samples for the next level computation and then write them back as in [8]. If a frame memory is not allowed, then most architectures implement some form of the line scan method. The architecture in [8] uses an internal buffer of size *BN*, where *B* is the height of the code block in a JPEG2000 architecture. The line-based implementation in [10] organizes the line buffer into two parts: the signal memory which stores both the low pass and the high pass outputs corresponding to the row-wise computations and the temporary memory which stores the temporary results for the column-wise computations. Details of the size of the memory as well as nature of memory accesses are described. A generic line-based implementation of an RPA-based approach has been presented in [24]. The important feature here is the efficient use of 2-port RAMs to implement the line buffers.

The 2D extension of the dual scan architecture (DSA) (see Section 3.7 for the 1D DSA) is also a line based implementation. Here two consecutive rows are scanned simultaneously in line scan order and the column processor computes on the row DWT coefficients. The row-processor in 2D DSA is identical to the 1D DSA presented in Section 3.7; registers are used to hold the even and odd pixels of each row and to interleave the input pairs of two consecutive rows. The internals of the column processor is the same as the row processor. However the 1 pixel registers in the row processors are replaced by 1-row delay units. Compared to conventional architectures, DSA needs roughly half of the time to compute lifting based 2-D DWT.

4.6. *Comparison of Performances*

A summary of the hardware and timing requirements of a few representative architectures is presented in

Fig. 24. The hardware complexity has been described in terms of datapath components and internal memory size. We list only the internal memory size since all the architectures require an external memory of size N^2 for input data of size $N \times N$. The timing performance has been compared with respect to the number of clock cycles to compute L levels of decomposition and the clock period.

Of the four architectures, the architecture in [15] has the smallest internal memory. This is because [15] is an RPA based approach that intersperses the computations at the higher levels with those of the lower levels. The architecture in [12], on the other hand, computes all the outputs of one level before starting the computations at the next level and has an internal memory of size $O(N^2)$. The datapath complexity of the architecture in [12] is by far the lowest.

The control complexity of the architecture in [15] is significantly higher than the others. This is because of the large number of control signals and switches that are used to organize the data before sending to the row and column computation units.

In terms of the timing performance, the architecture in [12] is pipelined and has the highest throughput ($1/T_m$). The architecture in [15, 16] requires the fewest number of cycles since they are RPA based, though the clock periods are significantly higher.

The architecture in [21] is specific to the (4,2) filters while the RPA concept that is applied to the architectures in [15, 16] can be applied to a large set of filters (not just (3,5), (9,7), Daub-4). The architecture in [12] is essentially a programmable architecture which supports implementation of a large set of filters on the same hardware platform.

5. Conclusions

In this paper, we presented a survey of the existing lifting based implementations of 1-dimensional and 2-dimensional Discrete Wavelet Transform. We briefly described the principles behind the lifting scheme in order to better understand the different implementation styles and structures. We have presented several architectures of different flavors ranging from highly parallel ones to highly folded ones to programmable ones. We provided a systematic derivation of each architecture and evaluated them with respect to their hardware and timing requirements.

References

1. S. Mallat, "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation," *IEEE Trans. Pattern Analysis And Machine Intelligence*, vol. 11, no. 7, 1989, pp. 674–693.
2. T. Acharya and P. S. Tsai, *JPEG2000 Standard for Image Compression: Concepts, Algorithms and VLSI Architectures.* John Wiley & Sons, Hoboken, New Jersey, 2004.
3. W. Sweldens, "The Lifting Scheme: A Custom-Design Construction of Biorthogonal Wavelets," *Applied and Computational Harmonic Analysis*, vol. 3, no. 15, 1996, pp. 186–200.
4. I. Daubechies and W. Sweldens, "Factoring Wavelet Transforms into Lifting Schemes," *The J. of Fourier Analysis and Applications*, vol. 4, 1998, pp. 247–269.
5. M.D. Adams and F. Kossentini, "Reversible Integer-to-Integer Wavelet Transforms for Image Compression: Performance Evaluation and Analysis," *IEEE Trans. on Image Processing*, vol. 9, 2000, pp. 1010–1024.
6. C.C. Liu, Y.H. Shiau, and J.M. Jou, "Design and Implementation of a Progressive Image Coding Chip Based on the Lifted Wavelet Transform," in *Proc. of the 11th VLSI Design/CAD Symposium*, Taiwan, 2000.
7. J.M. Jou, Y.H. Shiau, and C.C. Liu, "Efficient VLSI Architectures for the Biorthogonal Wavelet Transform by Filter Bank and Lifting Scheme," in *IEEE International Symposium on Circuits and Systems*, Sydney, Australia, 2001, pp. 529–533.
8. C.J Lian, K.F. Chen, H.H. Chen, and L.G. Chen, "Lifting Based Discrete Wavelet Transform Architecture for JPEG2000," in *IEEE International Symposium on Circuits and Systems*, Sydney, Australia, 2001, pp. 445–448.
9. P.-Y. Chen, "VLSI Implementation for One-Dimensional Multilevel Lifting-Based Wavelet Transform," *IEEE Transactions on Computers*, vol. 53, no. 4, 2004.
10. W.H. Chang, Y.S. Lee, W.S. Peng, and C.Y. Lee, "A Line-Based, Memory Efficient and Programmable Architecture for 2D DWT Using Lifting Scheme," in *IEEE International Symposium on Circuits and Systems*, Sydney, Australia, 2001, pp. 330–333.
11. C.T. Huang, P.C. Tseng, and L.G. Chen, "Flipping Structure: An Efficient VLSI Architecture for Lifting-Based Discrete Wavelet Transform," in *IEEE Transactions on Signal Processing*, 2004, pp. 1080–1089.
12. K. Andra, C. Chakrabarti, and T. Acharya, "A VLSI Architecture for Lifting-Based Forward and Inverse Wavelet Transform," *IEEE Trans. of Signal Processing*, vol. 50, no. 4, 2002, pp. 966–977.
13. M. Vishwanath, "The Recursive Pyramid Algorithm for the Discrete Wavelet Transform," in *IEEE Transactions on Signal Processing*, vol. 42, 1994, pp. 673–676.
14. H. Liao, M.K. Mandal, and B.F. Cockburn, "Novel Architectures for Lifting-Based Discrete Wavelet Transform," in *Electronics Letters*, vol. 38, no. 18, 2002, pp. 1010–1012.
15. H. Liao, M.K. Mandal, and B.F. Cockburn, "Efficient Architectures for 1-D and 2-D Lifting-Based Wavelet Transform," *IEEE Transactions on Signal Processing*, vol. 52, no. 5, 2004, pp. 1315–1326.
16. G.C. Jung, D.Y. Jin, and S.M. Park, "An Efficient Line Based VLSI Architecture for 2-D Lifting DWT," in *The 47th IEEE International Midwest Symposium on Circuits and Systems*, 2004.

17. M. Martina, G. Masera, G. Piccinini, and M. Zamboni, "Novel JPEG 2000 Compliant DWT and IWT VLSI Implementations," *Journal of VLSI Signal Processing*, vol. 34, 2003, pp. 137–153.
18. C. Chrysafis and A. Ortega, "Line-Based, Reduced Memory, Wavelet Image Compression," *IEEE Trans. on Image Processing*, vol. 9, no. 3, 2000, pp. 378–389.
19. J. Reichel, M. Nadenau, and M. Kunt, "Row-Based Wavelet Decomposition Using the Lifting Scheme," *Proceedings of the Workshop on Wavelet Transforms and Filter Banks*, Brandenburg an der Havel, Germany, March 5–7, 1999.
20. W. Jiang and A. Ortega, "Lifting Factorization-Based Discrete Wavelet Transform Architecture Design," *IEEE Trans, on Circuits and Systems for Video Technology*, vol. 11, 2001, pp. 651–657.
21. M. Ferretti and D. Rizzo, "A Parallel Architecture for the 2-D Discrete Wavelet Transform with Integer Lifting Scheme," *Journal of VLSI Signal Processing*, vol. 28, 2001, pp. 165–185.
22. M.Y. Chiu, K.-B. Lee, and C.-W. Jen, "Optimal Data Transfer and Buffering Schemes for JPEG 20000 Encoder," in *Proceedings of the IEEE Workshop on Design and Implementation of Signal Processing Systems*, 2003, pp. 177–182.
23. C.-T. Huang, P.-C. Tseng, and L.-G. Chen, "Memory Analysis and Architecture for Two-Dimensional Discrete Wavelet Transform," in *Proceedings of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 2004, pp. V13–V16.
24. P.-C. Tseng, C.-T. Huang, and L.-G. Chen, "Generic RAM-Based Architecture for Two-Dimensional Discrete Wavelet Transform with Line-Based Method," in *Proceedings of the Asia-Pacific Conference on Circuits and Systems*, 2002, pp. 363–366.

Tinku Acharya received his B.Sc. (Honors) in Physics, B.Tech. and M.Tech. in Computer Science from University of Calcutta, India, and the Ph.D. in Computer Science from University of Central Florida, USA, in 1984, 1987, 1989, and 1994, respectively. He is currently the Chief Technology Officer of Avisere Inc., Tucson, Arizona, USA. Dr. Acharya is also an Adjunct Professor in the Department of Electrical Engineering, Arizona State University, Tempe, USA.

Before joining Avisere, Dr. Acharya served in Intel Corporation (1996–2002), where he led several R&D teams toward development of algorithms and architectures in image and video processing, multimedia computing, PC-based digital camera, reprographics architecture for color photo-copiers, 3G cellular telephony, analysis of next-generation microprocessor architecture, etc. Before Intel, Dr. Acharya was a consulting engineer at AT&T Bell Laboratories (1995–1996), a research faculty at the Institute of Systems Research, Institute of Advanced Computer Studies, University of Maryland at College Park (1994–1995), and held visiting faculty positions at Indian Institute of Technology, Kharagpur. He served as Systems Analyst in National Informatics Center, Planning Commission, Government of India (1988–1990). He collaborated in research and development with Xerox Palo Alto Research Center (PARC), Eastman Kodak Corporation, and many other institutions worldwide.

Dr. Acharya is inventor of 88 US patents and 14 European patents. He authored over 80 technical papers and four books—Image Processing: Principles and Applications (Wiley, New Jersey, 2005), JPEG2000 Standard for Image Compression: Concepts, Algorithms, and VLSI Architectures (Wiley, 2004), Information Technology: Principles and Applications (Prentice-Hall India, 2004), and Data Mining: Multimedia, Soft Computing and Bioinformatics (Wiley, 2003).

Dr. Acharya is a Fellow of the National Academy of Engineers (India), Life Fellow of the Institution of Electronics and Telecommunication Engineers (FIETE), and Senior Member of IEEE. His current research interests are in computer vision, image processing, multimedia data mining, bioinformatics, and VLSI architectures and algorithms.
tinku_acharya@ieee.org

Chaitali Chakrabarti received the B.Tech. degree in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur, India in 1984, and the M.S. and Ph.D degrees in electrical engineering from the University of Maryland at College Park, USA, in 1986 and 1990 respectively. Since August 1990, she has been with the Department of Electrical Engineering, Arizona State University, Tempe, where she is now a Professor. Her research interests are in the areas of low power embedded systems design including memory optimization, high level synthesis and compilation, and VLSI architectures and algorithms for signal processing, image processing and communications.

Dr. Chakrabarti is a member of the Center for Low Power Electronics, the Consortium for Embedded Systems and Connection One. She received the Research Initiation Award from the National Science Foundation in 1993, a Best Teacher Award from the College of Engineering and Applied Sciences, ASU, in 1994, and the Outstanding Educator Award from the IEEE Phoenix section in 2001. She has served on the program committees of ICASSP, ISCAS, SIPS, ISLPED and DAC. She is currently an Associate Editor of the IEEE Transactions on Signal Processing and the Journal of VLSI Signal Processing Systems. She is also the TC Chair of the sub-committee on Design and Implementation of Signal Processing Systems, IEEE Signal Processing Society.
chaitali@asu.edu