A VLSI Architecture for Lifting-Based Forward and Inverse Wavelet Transform

Kishore Andra, Chaitali Chakrabarti, *Member, IEEE*, and Tinku Acharya, *Senior Member, IEEE*

***Abstract*—In this paper, we propose an architecture that performs the forward and inverse discrete wavelet transform (DWT) using a lifting-based scheme for the set of seven filters proposed in JPEG2000. The architecture consists of two row processors, two column processors, and two memory modules. Each processor contains two adders, one multiplier, and one shifter. The precision of the multipliers and adders has been determined using extensive simulation. Each memory module consists of four banks in order to support the high computational bandwidth. The architecture has been designed to generate an output every cycle for the JPEG2000 default filters. The schedules have been generated by hand and the corresponding timings listed. Finally, the architecture has been implemented in behavioral VHDL. The estimated area of the proposed architecture in 0.18-μ technology is 2.8 mm square, and the estimated frequency of operation is 200 Mhz.**

***Index Terms*—JPEG 2000, lifting, VLSI architectures, wavelet transform.**

I. Introduction

THE discrete wavelet transform (DWT) is being increasingly used for image coding. This is due to the fact that DWT supports features like progressive image transmission (by quality, by resolution), ease of compressed image manipulation, region of interest coding, etc. DWT has traditionally been implemented by convolution. Such an implementation demands both a large number of computations and a large storage—features that are not desirable for either high-speed or low-power applications. Recently, a lifting-based scheme that often requires far fewer computations has been proposed for the DWT [1], [2].

The main feature of the lifting based DWT scheme is to break up the highpass and lowpass filters into a sequence of upper and lower triangular matrices and convert the filter implementation into banded matrix multiplications [1], [2]. Such a scheme has several advantages, including "in-place" computation of the DWT, integer-to-integer wavelet transform (IWT), symmetric forward and inverse transform, etc. Therefore, it comes as no surprise that lifting has been chosen in the upcoming JPEG2000 standard [3].

In the JPEG2000 verification model (VM) Version 8.5 [4], the following wavelet filters have been proposed: (5, 3) (the highpass filter has five taps and the lowpass filter has three taps), (9, 7), C(13, 7), S(13, 7), (2, 6), (2, 10), and (6, 10). To be JPEG2000 compliant, the coder should be able to at least provide a (5, 3) filter in lossless mode and a (9, 7) filter in lossy mode. In this paper, we propose a unified architecture capable of executing all the filters mentioned above using the lifting scheme. Since different filters have different computational requirements, we focus on the configuration that ensures an output in every cycle for the JPEG2000 part I default filters. The proposed architecture computes multilevel DWT for both the forward and the inverse transforms, one level at a time, in a row-column fashion. There are two row processors to compute along the rows and two column processors to compute along the columns. While this arrangement is suitable or filters that require two banded-matrix multiplications [e.g., (5, 3) wavelet], filters that require four banded-matrix multiplications [e.g., (9, 7) wavelet] require all four processors to compute along the rows or along the columns. The outputs generated by the row and column processors (that are used for further computations) are stored in memory modules. The memory modules are divided into multiple banks to accommodate high computational bandwidth requirements. The architecture has been simulated using behavioral VHDL and the results compared with C code implementation. The proposed architecture is an extension of the architecture for the forward transform that was presented in [5].

A number of architectures have been proposed for calculation of the convolution-based DWT [6]–[11]. The architectures are mostly folded and can be broadly classified into serial architectures (where the inputs are supplied to the filters in a serial manner) and parallel architectures (where the inputs are supplied to the filters in a parallel manner). The serial architectures are either based on systolic arrays that interleave the computation of outputs of different levels to reduce storage and latency [6]–[8] or on digit pipelining, which implements the filterbank structure efficiently [9], [10]. The parallel architectures implement interleaving of the outputs and support pipelining to any level [11].

Recently, a methodology for implementing lifting-based DWT that reduces the memory requirements and communication between the processors, when the image is broken up into blocks, has been proposed in [12]. An architecture to perform lifting based DWT with (5, 3) filter that uses interleaving has been proposed in [13]. For a system that consists of the lifting-based DWT transform followed by an embedded zero-tree algorithm, a new interleaving scheme that reduces the number of memory accesses has been proposed in [14]. Finally, a lifting-based DWT architecture capable of performing filters with one lifting step, i.e., one predict and one update step, is presented in [15]. The outputs are generated in an interleaved fashion. The datapath is not pipelined, resulting in a large clock

Manuscript received November 20, 2000; revised January 7, 2002. The associate editor coordinating the review of this paper and approving it for publication was Dr. Edwin Hsing-Men Sha.

K. Andra and C. Chakrabarti are with the Department of Electrical Engineering, Telecommunications Research Center, Arizona State University, Tempe, AZ 85287-5706 USA (e-mail: kishore@asu.edu; chaitali@asu.edu).

T. Acharya is with Intel Corporation, Tempe, AZ 85226 (e-mail: tinku.acharya@intel.com).

Publisher Item Identifier S 1053-587X(02)02386-3.

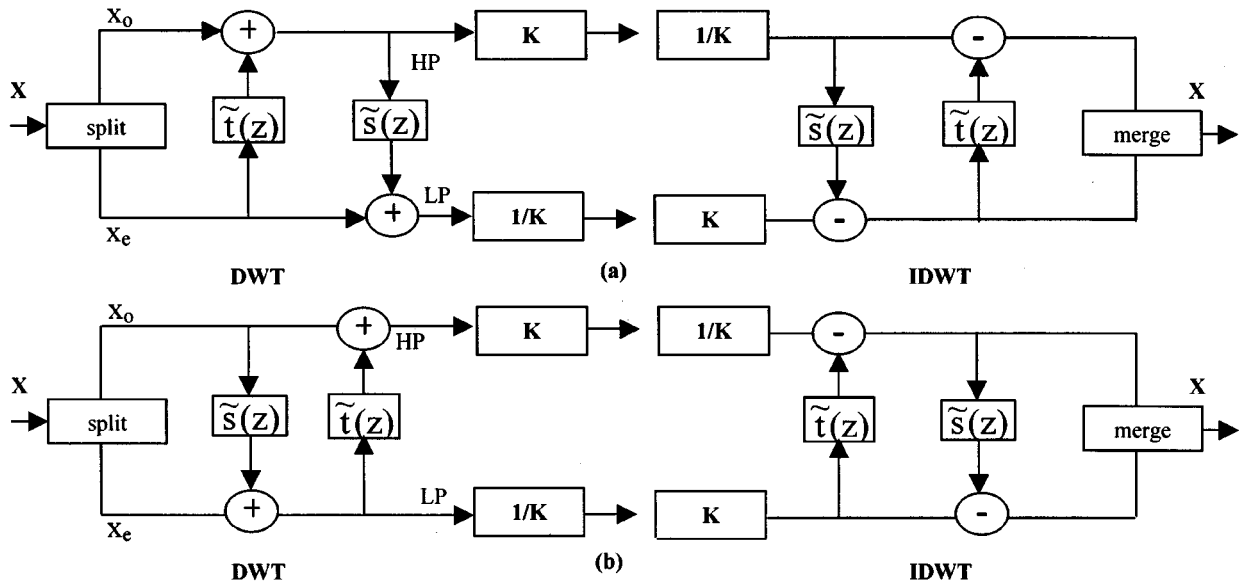

Fig. 1. Lifting Schemes. (a) Scheme 1. (b) Scheme 2.

period. In contrast, the proposed four processor architecture can perform transforms with one or two lifting steps one level at a time. Interleaving is not done since the entropy coder of JPEG2000 performs the coding in a intra-subband fashion (coefficients in higher levels are not required along with the first level coefficients). Furthermore, the data path is pipelined, and the clock period is determined by the memory access time.

The rest of the paper is organized as follows. In Section II, we give a brief overview of the lifting scheme. Precision analysis has been conducted for all the filters in Section III. The proposed architecture, including the memory organization and the control structure, are explained in Section IV. The timing performance of the architecture is discussed in Section V. The implementation details are presented in Section VI. The paper is concluded in Section VII. The lifting matrices for the filters are included in the Appendix.

II. Lifting-Based DWT

The basic principle of the lifting scheme is to factorize the polyphase matrix of a wavelet filter into a sequence of alternating upper and lower triangular matrices and a diagonal matrix [1], [2]. This leads to the wavelet implementation by means of banded-matrix multiplications.

Let $\tilde{h}(z)$ *and* $\tilde{g}(z)$ be the lowpass and highpass analysis filters, and let $h(z)$ *and* $g(z)$ be the lowpass and highpass synthesis filters. The corresponding polyphase matrices are defined as

$$\tilde{P}(z)=\begin{bmatrix}\tilde{h}_e(z) & \tilde{h}_o(z)\\ \tilde{g}_e(z) & \tilde{g}_o(z)\end{bmatrix} \quad \text{and} \quad P(z)=\begin{bmatrix}h_e(z) & g_e(z)\\ h_o(z) & g_o(z)\end{bmatrix}.$$

It has been shown in [1] and [2] that if $(\tilde{h},\tilde{g})$ is a complementary filter pair, then $\tilde{P}(z)$ can always be factored into lifting steps as

$$\tilde{P}_1(z)=\begin{bmatrix}K & 0\\ 0 & \frac{1}{K}\end{bmatrix}\prod_{i=1}^{m}\begin{bmatrix}1 & \tilde{s}_i(z)\\ 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\ \tilde{t}_i(z) & 1\end{bmatrix} \quad \text{or}$$

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

where K is a constant. The two types of lifting schemes are shown in Fig. 1.

Scheme 1 [see Fig. 1(a)], which corresponds to the $\tilde{P}_1(z)$ factorization, consists of three steps:

1) *Predict* step, where the even samples are multiplied by the time domain equivalent of $\tilde{t}(z)$ and are added to the odd samples;
2) *Update* step, where updated odd samples are multiplied by the time domain equivalent of $\tilde{s}(z)$ and are added to the even samples;
3) *Scaling* step, where the even samples are multiplied by $1/K$ and odd samples by K.

The inverse DWT is obtained by traversing in the reverse direction, changing the factor K to $1/K$, factor $1/K$ to K, and reversing the signs of coefficients in $\tilde{t}(z)$ and $\tilde{s}(z)$.

In Scheme 2 [see Fig. 1(b)], which corresponds to the $\tilde{P}_2(z)$ factorization, the odd samples are calculated in the first step, and the even samples are calculated in the second step. The inverse is obtained by traversing in the reverse direction.

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 the integer wavelet transform (IWT) [16]. 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 [16] to split the scaling step into additional lifting steps to achieve IWT. We do not explore this option.

Example: Let us consider the (5, 3) filter, with the following filter coefficients:

Highpass: $(-1/8, 2/8, 6/8, 2/8, -1/8)$;
Lowpass: $(-1/2, 1, -1/2)$.

The polyphase matrix of the above filter is

$$\tilde{P}_1(z)=\begin{bmatrix}-\frac{1}{8}z+\frac{6}{8}-\frac{1}{8}z^{-1} & \frac{2}{8}+\frac{2}{8}z\\ -\frac{1}{2}-\frac{1}{2}z^{-1} & 1\end{bmatrix}.$$

A possible factorization of $\tilde{P}_1(z)$, which leads to a band matrix multiplication (in the time domain), is

$$\tilde{P}_1(z) = \begin{bmatrix} 1 & 0.25(1+z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -0.5(1+z^{-1}) & 1 \end{bmatrix}.$$

If the signal is numbered from 0 and if even terms are considered to be the lowpass values and the odd terms the highpass values, we can interpret the above matrices in the time domain as

$$y_{2i+1} = -0.5(x_{2i} + x_{2i+2}) + x_{2i+1};$$
$$y_{2i} = 0.25(y_{2i+1} + y_{2i+3}) + x_{2i} \quad \text{where } 0 \le i < N/2$$

where xs are the signal values, and ys are the transformed signal values. Note that the odd samples are calculated from even samples, and even samples are calculated from the updated odd samples. The corresponding matrices M_1 *and* M_2 are shown in the following. Here, $a = -0.5$, and $b = 0.25$.

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

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

The transform of the signal X is $Y = X * M_1 * M_2$, whereas the inverse is $X = Y * M_2 * M_1$. ■

In this work, we have considered a block wavelet transform with a single sample overlap wavelet transform (SSOWT), as recommended in JPEG2000 VM [4]. As a result, the number of elements in a row or a column is odd. In addition, the first and last values in the input signal do not change on applying the transform. In JPEG2000 Part I [3], symmetric extension is suggested to be performed at the boundaries, and in JPEG2000 Part II [3], a slightly different definition of SSOWT is used. However, both of these cases can be easily handled with minimal changes to address the generation scheme in the proposed architecture. In this paper, we discuss all the details of the architecture based on the VM definition of the SSOWT.

1) Classification of Filters: We classify the wavelet filters based on the number of factorization matrices: A two-matrix factorization, corresponding to one predict and one update step, is denoted by $2M$, and a four-matrix factorization, corresponding to two predict steps and two update steps, is denoted by $4M$. The wavelet filters (5, 3), C(13, 7), S(13, 7), (2, 6), (2, 10) correspond to $2M$, whereas filters (9, 7) and (6, 10) correspond to $4M$. Furthermore, filters (5, 3), C(13, 7), S(13, 7), and (9, 7) use lifting Scheme 1 [see Fig. 1(a)], whereas (2, 6), (2, 10), and (6, 10) use lifting Scheme 2 [see Fig. 1(b)]. Filters (2, 6), (2, 10), (9, 7), and (6, 10) require a scaling step. The factorization matrices for the seven filters are given in the Appendix. The width of the band of the matrices for the various filters is given in Table I. The wider the band, the higher the number of computations, and the higher the amount of storage that is required for the intermediate results.

TABLE I
WIDTHS OF THE BANDS IN THE MATRICES

Filter	M1	M2	M3	M4	Scaling
(5,3)	3	3	-	-	No
C(13,7)	7	7	-	-	No
S(13,7)	7	7	-	-	No
(2,6)	2	5	-	-	Yes
(2,10)	2	9	-	-	Yes
(9,7)	3	3	3	3	Yes
(6,10)	2	4	4	5	Yes

TABLE II
COMPUTATIONAL COMPLEXITY COMPARISON BETWEEN CONVOLUTION AND LIFTING-BASED SCHEMES FOR A HIGHPASS, LOWPASS PAIR

Filter	Multiplications / Shifts		Additions	
	Convolution	Lifting	Convolution	Lifting
(5,3)	4	2	6	4
C(13,7)	8	4	14	8
S(13,7)	8	4	14	8
(2,6)	2	3	6	4
(2,10)	3	4	10	6
(9,7)	9	5	14	8
(6,10)	8	8	14	8

2) Comparison With Convolution: The number of computations required for calculation of a highpass, lowpass pair of wavelet transforms using convolution and lifting scheme is given in Table II. The reduction in the number of multiplications for the lifting scheme is significant for odd-tap filters compared with convolution. For even-tap filters, the convolution scheme has fewer or an equal number of multiplications. The number of additions is lower for lifting in both odd and even tap filters. Such reductions in the computational complexity makes lifting-based schemes attractive for both high throughput and low-power applications.

III. PRECISION ANALYSIS

We have carried out a comparison study between the floating-point and the fixed-point implementations (using C) to determine the number of bits required for "satisfactory" lossy and lossless performance in the fixed-point implementation. We have used three gray-scale images—baboon, barbara, and fish—each of size 513×513, with 8-bit pixels and carried out the study for five levels of decomposition. The results are validated with 15 gray scale images (8-bit pixels) from USC-SIPI database [17] (Images-5.2.08–10, 7.1.01–04, 7.1.06–10, boat, elaine, ruler, and gray21 from the Miscellaneous directory).

A. Filter Coefficients

The filter coefficients for the seven filters considered range from 0.003 906 to 2. In order to convert the filter coefficients to integers, the coefficients are multiplied with 256 (i.e., shifted left by 8 bits). The range of the coefficients is now 1 to 512, which implies that the coefficients require 10 bits to be represented in 2's complement form. At the end of the multiplication, the product is shifted right by 8 to get the required result. This is implemented in hardware by rounding the eight least significant bits. The products are rounded to the next highest integer. For instance, numbers ≥ 965.50 are rounded to 966, and numbers <965.5 are rounded to 965. It should be noted that instead of applying rounding on the result of the filter operation (which results in bigger accumulators) as in [16], rounding is applied to the individual product terms.

B. Signal Values

The signal values have to be shifted left as well in order to increase the precision; the extent of the shift is determined using image quality analysis. In order to experiment with shifts ranging from 0 to 5 bits, we introduce additional bits (ABs). In conventional fixed-point filter implementation, instead of shifting the input samples, the coefficients are shifted appropriately. This method cannot be directly applied to lifting-based filter implementation. Consider the general structure in lifting-based schemes

$$y = a(x1 + x2) + b(x3 + x4) + x5$$

where a *and* b are the filter coefficients, xs are the signal samples, and y is the transform value. We observe that since $x5$ has a coefficient of 1, if the filter coefficients are shifted by extra bits, a shifting operation has to be performed on the $x5$ term to maintain the data alignment. To avoid this, the signal values are shifted at the input.

Example: Consider the general structure in a lifting-based scheme with $a = 0.2345, b = 1.4567, x1 = 24, x2 = 43, x3 = 156, x4 = 56$, and $x5 = 10$. The floating-point implementation result is $y = 334.5319$. Let us assume that coefficients are shifted left by 8 bits (and rounded to nearest integer) and number of ABs $= 2$. Then, $a = 60, b = 373, x1 = 96, x2 = 172, x3 = 624, x4 = 224$, and $x5 = 40$. The products are $60(96+172) = 16\,080$ and $373(624 + 224) = 316\,304$. Shifting the product right by 8 bits and rounding will yield 63 and 1236. Therefore, $y = 63 + 1236 + 40 = 1339$. This should be interpreted as round$[1339/4 +$ decimal equivalent of two LSBs of$1139] =$ round$[334 + 0.75] = 335$.

C. Results

All through this work, we define SNR as

$$\text{SNR (dB)} = 20\log_{10}\left(\frac{\sum|\text{Signal}|}{\sum|\text{Signal} - \text{fixed point data}|}\right)$$

where "Signal" corresponds to the original image data.

The SNR values, for the baboon image, after five levels of forward and inverse transform with truncation and rounding, are given in Tables III and IV, respectively. Filters (2, 6)L and (2, 10)L are scaling step-free factorizations of (2, 6) and (2, 10) filters given in [18]. Finally, even though the lifting coefficients for (5, 3) and (2, 6)L filters are multiples of 2 and can be implemented using shift operations, we have used multiplications in this analysis for comparison purposes.

TABLE III
SNR VALUES AFTER FIVE LEVELS OF DWT WITH TRUNCATION FOR BABOON IMAGE

ABs	(5,3)	C(13,7)	S(13,7)	(2,6)L	(2,10)L	(2,6)	(2,10)	(6,10)	(9,7)
0	12.98	3.99	1.78	18.80	18.15	19.57	17.61	0.99	-0.72
1	19.61	10.15	7.94	26.33	25.11	26.16	23.75	7.08	5.38
2	25.81	16.17	13.92	32.68	31.62	32.56	29.84	13.11	11.36
3	32.79	22.24	20.01	39.82	38.79	39.64	36.15	19.01	17.36
4	36.96	28.01	26.32	70.64	48.55	52.03	44.31	24.99	23.33
5	Inf	33.09	32.89	Inf	99.81	77.28	65.78	31.17	29.15

TABLE IV
SNR VALUES AFTER FIVE LEVELS OF DWT FOR WITH ROUNDING FOR BABOON IMAGE

ABs	(5,3)	C(13,7)	S(13,7)	(2,6)L	(2,10)L	(2,6)	(2,10)	(6,10)	(9,7)
0	18.94	27.39	27.09	18.97	18.98	23.43	23.01	32.65	25.77
1	24.40	31.04	30.85	24.52	24.56	28.71	28.49	37.86	31.62
2	30.20	37.47	36.48	31.24	31.32	35.51	35.39	44.54	38.37
3	34.50	48.72	48.21	36.82	36.80	50.13	45.64	48.91	47.31
4	77.46	113.2	110.3	73.63	70.89	67.57	67.93	56.54	67.39
5	Inf	Inf	Inf	Inf	Inf	78.08	78.52	60.01	85.75

From the tables, we see that for (5, 3) and (2, 6)L filters to obtain lossless performance, truncation with five ABs is sufficient, but for the rest of the filters, which can attain lossless performance, rounding is required. In case of lossy filters, such as (2, 6) and (2, 10) filters, rounding does not improve the performance significantly, but for (6, 10) and (9, 7) filters, rounding improves performance by 30 dB. Based on these observations, we conclude that rounding is essential for better performance.

From Table IV, we also conclude that for lossless performance, five ABs are required. To determine the number of ABs required for lossy performance, we have to consider two cases: implicit quantization and explicit quantization. In the first case, the DWT coder is followed by a lossless entropy coder; therefore, the required quantization is performed by controlling the precision of the DWT coefficients. If this is the case, then two ABs are sufficient to obtain "satisfactory" performance with ~ 35 dB SNR. In the second case, the DWT coder is followed by a explicit quantizer, which is followed by a lossless entropy coder as in JPEG2000. In this case, five ABs are required to obtain the best possible SNR performance as the quantization would introduce substantial loss in SNR.

Once the number of ABs are fixed, we need to determine the width of the data path. This can be done by observing the maximum/minimum values for the transformed values at the end of each level of decomposition and taking the largest/smallest among them. The maximum and minimum values for the baboon, barbara, fish, and ruler images with ABs $= 5$ are given in Table V.

From Table V, we see that 16 bits are required to represent the transform values (in 2's complement representation). It should be noted that values in Table V are obtained at the end of the filtering operation, but the individual products can be greater than the final values. Indeed, this is the case for few of the coefficients in case of ruler image using the (9, 7) filter. In such

TABLE V
MAXIMUM AND MINIMUM VALUES WITH ABs = 5

	baboon		barbara	
Filter	Max.	Min.	Max.	Min.
(5,3)	8642	-7611	5950	-5296
C(13,7)	7216	-8487	6269	-5837
S(13,7)	9226	-11008	7429	-8705
(2,6)L	6250	-6183	5930	-5036
(2,10)L	6287	-5951	5778	-5261
(2,6)	11136	-8946	17006	-5913
(2,10)	9066	-8376	12484	-7256
(6,10)	7692	-7451	7515	-6124
(9,7)	5983	-6817	6392	-5579

	fish		ruler	
Filter	Max.	Min.	Max.	Min.
(5,3)	4876	-5253	10774	-14821
C(13,7)	4556	-5171	16986	-18634
S(13,7)	5664	-5891	12428	-15983
(2,6)L	4147	-4096	16575	-16575
(2,10)L	3887	-4097	16986	-16987
(2,6)	5126	-11308	16576	-29022
(2,10)	3887	-7516	16986	-18634
(6,10)	6424	-5221	12007	-13220
(9,7)	3549	-4870	12713	-17207

cases, the product is saturated at 16 bits. As the occurrences of such coefficients are very limited, the SNR performance is not affected. Using similar analysis, it was found that 13 bits of precision is required when ABs = 2.

Based on these observations, in our architecture, the data path width is fixed at 16 bits. The adders and shifters are designed for 16-bit data. The multiplier multiplies a 16-bit number (signal value) by a 10-bit number (filter coefficient) and then rounds the product with eight LSBs (to account for the increased precision of the filter coefficients) and two MSBs (16 bits are required to represent the outputs and therefore, the two MSBs would be sign extension bits) to form a 16-bit output.

IV. PROPOSED VLSI ARCHITECTURE

The proposed architecture calculates the forward transform (DWT) and the inverse transform (IDWT) in row-column fashion on a block of data of size $N \times N$. To perform the DWT, the architecture reads in the block of data, carries out the transform, and outputs the LH, HL, and HH data at each level of decomposition. The LL data is used for the next level of decomposition. To perform the IDWT, all the sub-bands from the lowest level are read in. At the end of the inverse transform, the LL values of the next higher level are obtained. The transform values of the three subbands (LH, HL, and HH) are read in, and the IDWT is carried out on the new data set.

The architecture, as shown in Fig. 2, consists of a row module (two row processors RP1 and RP2 along with a register file REG1), a column module (two column processors CP1, CP2 and a register file REG2), and two memory modules (MEM1, MEM2). As mentioned earlier, DWT and IDWT are symmetrical if the lifting scheme is used. Hence, in the rest of the paper, we discuss all the details in terms of DWT as an extension to IDWT is straightforward.

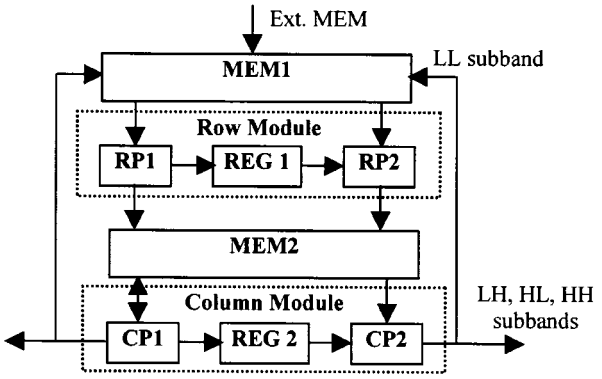

Fig. 2. Block diagram of the proposed architecture.

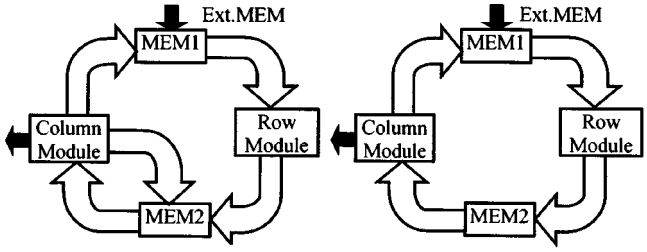

Fig. 3. Data flow for (a) $2M$ filters and (b) $4M$ filters.

A. Data Flow for 2M Filters

In the $2M$ case (i.e., when lifting is implemented by two factorization matrices), processors RP1 and RP2 read the data from MEM1, perform the DWT along the rows, and write the data into MEM2. Processor CP1 reads the data from MEM2, performs the column wise DWT along alternate *rows*, and writes the HH and LH subbands into MEM2 and Ext.MEM. Processor CP2 reads the data from MEM2, performs the column-wise DWT along the *rows* on which the CP1 did not work, and writes LL sub-band to MEM1 and HL sub-band to Ext.MEM. The data flow is shown in Fig. 3(a).

B. Data Flow for 4M Filters

In the $4M$ case (i.e., when lifting is implemented by four factorization matrices), there are *two passes* with transform along one dimension being calculated in a pass. In the first pass, RP1 and RP2 read in the data from MEM1, execute the first two matrix multiplications, and write the result into MEM2. CP1 and CP2 execute the next two matrix multiplications and write results (highpass and lowpass terms along the rows) to MEM2. This finishes the transform along rows. In the second pass, the transform is calculated along columns. At the end of the second pass, CP1 writes HH and LH sub-bands to Ext.MEM, whereas CP2 writes the LL sub-band to MEM1 and the HL sub-band to Ext.MEM. The data flow is shown in Fig. 3(b).

C. Transform Computation Style

In the $2M$ case, the latency and memory requirements would be very large if the column transform is started after finishing the row transform. To overcome this, the column processors also have to work row-wise. This is illustrated in Fig. 4 for the (5, 3) filter for a signal of length 5.

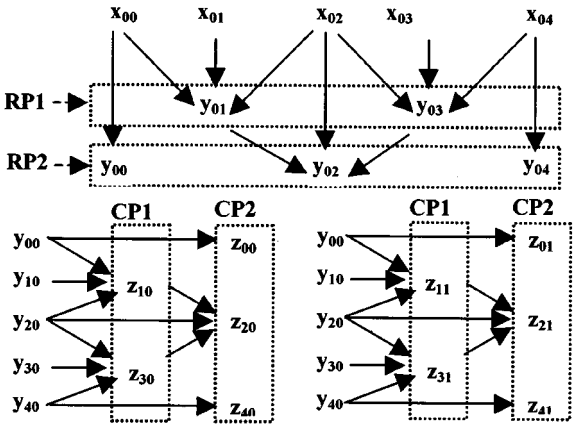

Fig. 4. Row and column processor data access patterns for the forward (5, 3) transform with $N = 5$.

RP1 calculates the *highpass* (odd) elements along the rows $y_{01}, y_{03}, \ldots$, etc., whereas RP2 calculates the *lowpass* (even) elements along the rows $y_{00}, y_{02}, y_{04}, \ldots$, etc. CP1 calculates the *highpass and lowpass* elements $z10, z11, \ldots; z30, z31, \ldots$, etc., along odd rows, and CP2 calculates *highpass and lowpass* elements $z00, z01, \ldots; z20, z21, \ldots; z40, z41, \ldots$, etc., along the even rows. Note that CP1 and CP2 start computations as soon as the required elements are generated by RP1 and RP2. This is further illustrated in the schedule given in Tables VIII and IX. In general, for $2M$ filters using Scheme 1 factorization, RP1 calculates the *highpass* values, and RP2 calculates the *lowpass* values along *all the rows*. CP1 and CP2 calculate both *highpass and lowpass* values along the *odd and even rows*, respectively. In case of Scheme 2 factorization, the roles of RP1 and RP2, as well as CP1 and CP2, are reversed.

In the case of $4M$ filters, all four processors calculate either the row or column transform at any given instant. In general, for $4M$ filters with Scheme 1 factorization, RP1 and CP1 calculate *highpass* values along the *rows* in the *first pass* and along *columns* in the *second pass*. Similarly RP2 and CP2 calculate *lowpass* values. As in the $2M$ case, for filters with Scheme 2 factorization, the roles of the processors are reversed.

D. Transform Computation Order

In the case of $2M$ filters, with the row and column processors working along the rows, the rows have to be calculated in a nonsequential fashion in order to minimize the size of the MEM2 module and to keep column processors active continuously. For example, in the (5, 3) filter, while performing row transform, the zeroth, second, and first elements of a row are required to update the first element (see Fig. 4). Therefore, while performing the column transform, the row transform of the zeroth row and the second row should have been completed before CP1 can start computations along the first row. The order in which the row processors and the column processors compute for a 9×9 block is described in Table VI.

Note that each filter needs a different order in which the row computations need to be finished. The order is determined by the factorization matrices. For instance, for the (5, 3) filter, the row processors calculate rows in the order 0, 2, 1, 4, 3, 6, 5, 8, 7 (see Table VI). CP1 starts computing along row 1 as soon as the first output from row 1 is available. After completing computation along row 1, CP1 starts computing along row 3, etc. CP2 starts after the first output from row 3 is available from CP1. It computes first along row 2, then along row 4, then row 6, etc. For $4M$ filters, sequential order of calculation is sufficient.

TABLE VI
ROW ORDER FOR PERFORMING THE TRANSFORM ON A 9×9 BLOCK

(5,3)			(13,7)			(2,6)			(2,10)		
Row	CP1	CP2	Row	CP1	CP2	Row	CP1	CP2	Row	CP1	CP2
0	-	-	0	-	-	0	*0*	-	0	-	-
2	-	-	2	-	-	1	-	-	1	*0*	-
1	1	*0*	4	-	-	3	2	-	3	-	-
4	-	-	1	1	-	2	-	1	2	2	-
3	3	2	6	-	*0*	5	-	-	5	-	-
6	-	-	3	3	-	4	4	3	4	4	1
5	5	4	8	-	-	7	-	-	7	-	-
8	-	-	5	5	2	6	6	5	6	6	3
7	7	6	-	-	-	-	-	-	-	-	-
-	-	-	7	7	4	8	*8*	*7*	8	*8*	5
-	-	*8*	-	-	-				-	-	-
			-	-	6				-	-	7
			-	-	-						
			-	-	*8*						

E. Row and Column Processor Design

Each filter requires a different configuration of adders, multipliers, and shifters in the data path in order to generate two coefficients (from different subbands) in every cycle. Table VII lists the number of data path components required for the filters under consideration. The (5, 3) filter requires two adders and a shifter in each processor and has the smallest requirement. The (13, 7) filter has the largest configuration (four adders and two multipliers) for RP1 and CP1, whereas filter (2, 10) has the largest configuration (five adders, two multipliers, and one shifter) for RP2 and CP2.

From Table VII, we see that 16 adders, eightmultipliers, and four shifters are needed in order for every filter to generate an output each clock cycle. However, if the data path did consist of these many resources, then for most filters, these resources would be grossly underutilized. This prompted us to look at a configuration that would generate two sub-band coefficients every clock cycle for the default JPEG2000 filters [(5, 3) and (9, 7) filters]. Such a configuration has fewer resources and is more heavily utilized. All four processors in the proposed architecture consist of two adders, one multiplier, and one shifter, as shown in Fig. 5. Since fewer resources are being used, two coefficients (from two subbands) are generated in *alternate* cycles for the (13, 7), (2, 10), and (6, 10) filters, whereas two coefficients are generated in *every* cycle for the (5, 3), (2, 6), and (9, 7) filters. Note that the MUXs at input have not been shown in Fig. 5. In order to carry out the scaling step, a shifter is connected to the output of the RP1 and RP2 processors, and a multiplier/shifter is connected to the output of the CP1 and CP2 processors.

TABLE VII
HARDWARE REQUIRED TO GENERATE AN OUTPUT EACH CLOCK CYCLE

Filter	Multipliers	Adders	Shifters
(5,3)	-	8	4
(13,7)	8	16	-
(2,6)	2	8	2
(2,10)	4	12	2
(9,7)	4	8	-
(6,10)	7	8	-

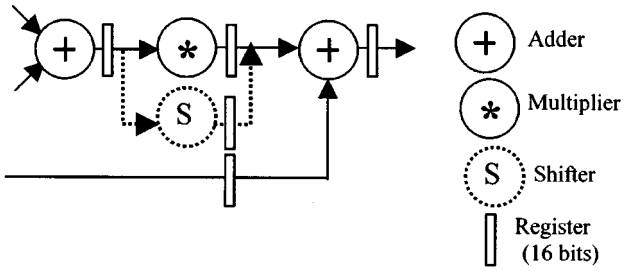

Fig. 5. Basic architecture of each processor.

F. Schedule

We have generated a detailed schedule for each of the filters by hand. The schedules are resource constrained list-based schedules, where the resources consist of an adder, a multiplier, and a shifter. It is assumed that the delay of the adder and shifter is one time unit and that the delay of the multiplier is four time units. This is justified since the multiplier is typically three times slower than an adder, and an additional addition operation is required to round the product. A snapshot of the schedule for the (5, 3) filter applied on a 9×9 block is provided in Tables VIII and IX.

The schedule in Table VIII should be read as follows. In the seventh cycle, Adder1 of RP1 adds the elements $(x_{2,0}, x_{2,2})$ and stores the sum in register RA1. The shifter (Shifter column) reads this sum in the next cycle (eighth cycle), carries out the required number of shifts (one right shift in this case as $a = -0.5$), and stores the data in register RS. The second adder (Adder2) reads the value in RS and subtracts the element $x_{2,1}$ to generate $y_{2,1}$ in the next cycle (ninth cycle). The output of the second adder is stored in a suitable memory location in MEM2 module and is also supplied to RP2 using REG1. Thus, to process a row of a 9×9 block, the RP1 processor takes four cycles. Adder 1 in RP2 starts computation in the sixth cycle. The gaps in the schedule for RP1 and RP2 are required to read the zeroth element of each row. Adder1 in CP1 starts in the 13th cycle to absorb the first element of row 1 computed by RP1 in the 14th cycle. Adder1 of CP2 starts after CP1 computes the first element in row 3 (25th cycle). The total time required to calculate an $N \times N$ block using the (5, 3) filter is $-2\lfloor N/2 \rfloor + 3Ta + 2Ts + N + 5 + \lfloor N/2 \rfloor N$ cycles, where Ta is the delay of an adder, and Ts is the delay of a shifter.

G. Memory

The proposed architecture consists of two memory modules: MEM1 and MEM2. The MEM1 module consists of two banks and MEM2 module consists of four banks. All the banks have one read and one write port. Further, we assume that two accesses/cycle are possible. The memory module structure is shown in Fig. 6.

TABLE VIII
PART OF THE SCHEDULE FOR RP1 AND RP2 FOR (5, 3) FILTER APPLIED ON A 9×9 BLOCK

Time	RP1			RP2		
	Adder1	Shifter	Adder2	Adder1	Shifter	Adder2
1	-	-	-	-	-	-
2	$x_{0,0}+x_{0,2}$	-	-	-	-	-
3	$x_{0,2}+x_{0,4}$	RA1	-	-	-	-
4	$x_{0,4}+x_{0,6}$	RA1	RS-$x_{0,1}$	-	-	-
5	$x_{0,6}+x_{0,8}$	RA1	RS-$x_{0,3}$	-	-	-
6	-	RA1	RS-$x_{0,5}$	$y_{0,1},y_{0,3}$	-	-
7	$x_{2,0}+x_{2,2}$	-	RS-$x_{0,7}$	$y_{0,3},y_{0,5}$	RA1	$y_{0,0}$
8	$x_{2,2}+x_{2,4}$	RA1	-	$y_{0,5},y_{0,7}$	RA1	RS+$x_{0,2}$
9	$x_{2,4}+x_{2,6}$	RA1	RS-$x_{2,1}$	-	RA1	RS+$x_{0,4}$
10	$x_{2,6}+x_{2,8}$	RA1	RS-$x_{2,3}$	-	-	RS+$x_{0,6}$
11	-	RA1	RS-$x_{2,5}$	$y_{2,1},y_{2,3}$	-	$y_{0,8}$
12	$x_{1,0}+x_{1,2}$	-	RS-$x_{2,7}$	$y_{2,3},y_{2,5}$	RA1	$y_{2,0}$
13	$x_{1,2}+x_{1,4}$	RA1	-	$y_{2,5},y_{2,7}$	RA1	RS+$x_{2,2}$
14	$x_{1,4}+x_{1,6}$	RA1	RS-$x_{1,1}$	-	RA1	RS+$x_{2,4}$

TABLE IX
PART OF THE SCHEDULE FOR CP1 AND CP2 FOR (5, 3) FILTER APPLIED ON AN 9×9 BLOCK

Time	CP1		
	Adder	Shifter	Adder
13	$y_{0,1}+y_{2,1}$	-	-
14	$y_{0,3}+y_{2,3}$	RA1	-
15	$y_{0,5}+y_{2,5}$	RA1	RS-$y_{1,1}$
16	$y_{0,7}+y_{2,7}$	RA1	RS-$y_{1,3}$
17	$y_{0,0}+y_{2,0}$	RA1	RS-$y_{1,5}$
18	$y_{0,2}+y_{2,2}$	RA1	RS-$y_{1,7}$
19	$y_{0,4}+y_{2,4}$	RA1	RS+$y_{1,0}$
20	$y_{0,6}+y_{2,6}$	RA1	RS+$y_{1,2}$
21	$y_{0,8}+y_{2,8}$	RA1	RS+$y_{1,4}$
22	$y_{2,1}+y_{4,1}$	RA1	RS+$y_{1,6}$
23	$y_{2,3}+y_{4,3}$	RA1	RS+$y_{1,8}$
24	$y_{2,3}+y_{4,3}$	RA1	RS-$y_{3,1}$

Time	CP2		
	Adder	Shifter	Adder
25	$z_{1,1}+z_{3,1}$	-	-
26	$z_{1,3}+z_{3,3}$	RA1	-
27	$z_{1,5}+z_{3,5}$	RA1	RS-$y_{2,1}$
28	$z_{1,7}+z_{3,7}$	RA1	RS-$y_{2,3}$
29	$z_{1,0}+z_{3,0}$	RA1	RS-$y_{2,5}$
30	$z_{1,2}+z_{3,2}$	RA1	RS-$y_{2,7}$
31	$z_{1,4}+z_{3,4}$	RA1	RS+$y_{2,0}$
32	$z_{1,6}+z_{3,6}$	RA1	RS+$y_{2,2}$
33	$z_{1,8}+z_{3,8}$	RA1	RS+$y_{2,4}$
34	$z_{3,1}+z_{5,1}$	RA1	RS+$y_{2,6}$
35	$z_{3,3}+z_{5,3}$	RA1	RS+$y_{2,8}$
36	$z_{3,3}+z_{5,3}$	RA1	RS-$y_{4,1}$

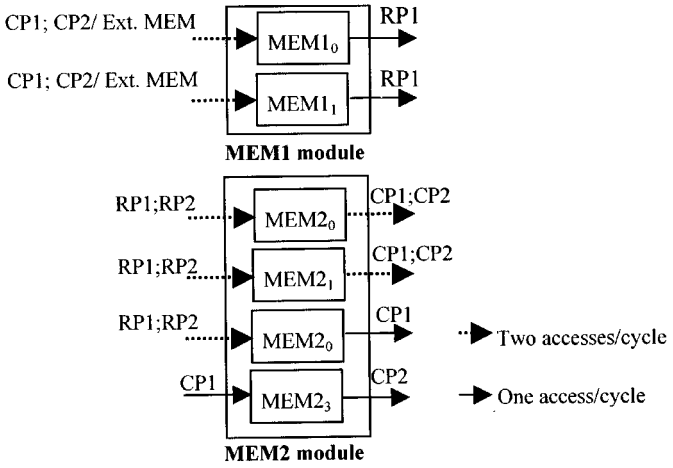

Fig. 6. Memory structure required for (5, 3) and (9, 7) filters.

1) Memory Organization:

MEM1 Module: The MEM1 module consists of two banks (MEM1_0 and MEM1_1), as shown in Fig. 6. Each bank contains either odd samples or even samples of a row. The data is stored into banks to minimize the number of ports needed. For example, in the case of the (5, 3) filter, MEM1_0 contains the odd samples, and MEM1_1 contains the even samples. Due to this arrangement, we need one read access for MEM1_0 to feed RP1 and two read accesses for MEM1_1 to feed RP1 and RP2. However, with additional registers, the even terms read by RP1 can be supplied to RP2, thereby decreasing the port requirement to one read port on MEM1_1. Both banks need one write port for Ext.MEM to write the raw input or for CP2 to write LL sub-band data at the end of each level. In the case of the (9, 7) filter, in the first pass, CP1 and CP2 write highpass and lowpass terms from the row transform to MEM1 simultaneously. Since dual access per cycle is possible, one write port on each bank is sufficient.

MEM2 Module: The MEM2 module consists of four banks (MEM2_0, MEM2_1, MEM2_2, and MEM2_3), as shown in Fig. 6. In the case of $2M$ filters, the banks contain a complete row of data. RP1 and RP2 write to the MEM2_0, MEM2_1, and MEM2_2 banks in a special order (see Table XI). These banks supply inputs to CP1 and CP2. CP1 writes to MEM2_3, and it is read by CP2. Four banks are required due to the nature of the calculation of the column transform along the rows. For example, during calculation of y_{11}, y_{13} and y_{12} using the (5, 3) filter (see Table VIII), two memory accesses are required by RP1: one for the even term and the other for the odd term. This is assuming there are two registers at the input of RP1, two registers at the input of RP2, and six registers for the even values required by RP2. On the other hand, consider calculation of column transform values z_{11}, z_{13}, and z_{12} (see Table IX). Here, $(y_{01}, y_{11}, y_{21} \rightarrow z_{11})$, $(y_{03}, y_{13}, y_{23} \rightarrow z_{13})$, and $(z_{11}, y_{12}, z_{13} \rightarrow z_{12})$. It can be seen that buffers at the input of RP1 are not useful, as a new row is accessed in every cycle. Therefore, all three inputs to CP1 have to be supplied by the MEM2 module. For CP2, one input can be buffered, but two inputs have to be supplied by MEM2. In conclusion, row processors need *two* inputs from the memory and *four* from the registers, whereas the column processors need *five* inputs from the memory and *one* input from a register. MEM2_2 and MEM2_3 supply two of the five inputs, and MEM2_0 and MEM2_1 supply the remaining three. Therefore, a dual read operation has to be performed on one of the banks: either MEM2_0 or MEM2_1. In the case of the (13, 7), (2, 6), and (2, 10) filters, a dual read operation is also required on the MEM2_3 bank.

TABLE X
NUMBER OF READ ACCESSES TO MEMORY AND REGISTERS TO GENERATE A PAIR OF LOWPASS AND HIGHPASS COEFFICIENTS

	Row processors		Column processors	
Filter	**MEM1**	**registers**	**MEM2**	**Registers**
(5,3)	2	4	5	1
(13,7)	3	7	9	1
(2,6)	3	3	5	1
(2,10)	3	5	7	1
(9,7)	2+2	4+4	2+2	4+4
(6,10)	2+2	3+3	2+2	5+5

In the case of $4M$ filters, only the MEM2_0 and MEM2_1 banks are used, and they contain either even or odd terms. RP1 writes to MEM2_0, and RP2 writes to MEM2_1. Both banks supply data to CP1. The data for CP2 is supplied through internal registers.

The number of memory and register *read* accesses for row processors and column processors to generate a highpass and a lowpass coefficient is given in Table X. Note that for the (13, 7) and (2, 10) filters, the accesses are spread over two cycles. For the (9, 7) and (6, 10) filters, accesses are spread over two passes. In the case of $2M$ filters, the row processors require two write accesses to the MEM2 module, whereas column processors require one write access to the MEM1 module. For $4M$ filters, row processors require two write accesses to the MEM2 module in both passes, whereas column processors require two write accesses in the first pass and one write access in the second pass, both to the MEM1 module.

2) Memory Size:

a) MEM1 Module: The memory banks in the MEM1 module read in the whole block in the beginning during the forward transform and read in the whole block at the last level during the inverse transform. Therefore, the memory banks are of size $N \times \lceil N/2 \rceil$ each.

b) MEM2 Module: As mentioned earlier, the $2M$ filters need four banks of memory in the MEM2 module. We can determine the size of the memory required in each of the banks based on when a particular bank is being updated and when the row data present in that bank is being used by CP1 or CP2. In other words, the size of the memory is a function of the lifetime of a row of data. For example, consider the (5, 3) filter. The order in which the rows are calculated is given in Table VI, and the order in which these rows are written into the MEM2 banks is given in Table XI.

In Table XI, x^{RP} indicates the transform of row x generated by the RP1 and RP2 processors. Similarly, x^{CP1} indicates the column-wise transform generated along the row x by CP1. The table can be read as follows: Data of 0^{RP} is written into MEM2_0, data of 2^{RP} into MEM2_1, and data of 1^{RP} into MEM2_2. CP1 uses the data from all these three banks, calculates 1^{CP1}, and

TABLE XI
PATTERN IN WHICH DATA IS WRITTEN INTO MEM2 BANKS FOR FORWARD (5, 3) FILTER

$MEM2_0$	$MEM2_1$	$MEM2_2$	$MEM2_3$
0^{RP}	-	-	-
-	2^{RP}	-	-
-	-	1^{RP}	1^{CP1}
4^{RP}	-	-	-
-	-	3^{RP}	3^{CP1}
-	6^{RP}	-	-
-	-	5^{RP}	5^{CP1}
8^{RP}	-	-	-
-	-	7^{RP}	7^{CP1}

writes into $MEM2_3$ and to Ext.MEM. Once the data from 3^{CP1} is available, CP2 calculates 2^{CP2} using 1^{CP1}, 3^{CP1} and 2^{RP} and writes the LL subband data to MEM1 and HL subband data to Ext.MEM. It can be observed from Table XI that the data available in a bank is used up before the next row of data is written into it. Therefore, it can be concluded that one row of data is required in each of the banks.

For the $4M$ filters, the size of the two banks $MEM2_0$ and $MEM2_1$ can be estimated from the maximum of the difference of the latencies between the RP1 and CP1 processors and the RP2 and CP2 processors. The total memory required for the filters is given in Table XII. For $Ta = Ts = 1$ and $Tm = 4$, the (9, 7) filter requires 17 elements to be stored in the banks $MEM2_0$ and $MEM2_1$. In contrast, the (5, 3) filter requires an entire row to be stored in all the four MEM2 banks.

H. Register Files

We need register files between the processors to minimize the number of memory accesses (as explained in previous section). The outputs from RP1 are stored in REG1 and are used by RP2. Similarly, REG2 acts as buffer between CP1 and CP2. For (2, 6) and (2, 10) filters, a partial sum has to be held for a time proportional to the multiplier delay. Table XIII lists the number of registers required for all the filters with $Ta = Ts = 1$ and $Tm = 4$.

I. Control

Control signals are needed primarily to maintain the steady flow of data to and from the processors. Our design consists of local controllers in each of the processors, which communicate with each other by hand shaking signals. Each local controller consists of three components

1) counter;
2) memory signal generation unit;
3) address generation unit.

Counter: Counters keep track of the number of rows and the number of elements in each row that have been processed. They are primarily used to generate the memory read and write signals. All the counters are capable of counting up to a maximum of $\lceil N/2 \rceil$.

Memory Read and Write Signals Generation Logic: The logic required for memory reads is driven by the counter output (i.e., row, element values). One of the inputs to the second adder (in all the processors) has to be read from memory, and the memory write signals are generated based on this signal.

Address Generation Unit: For MEM1 module, an "in place" addressing scheme is required in case of both $2M$ and $4M$ filters. Note that if a simple addressing scheme (ex. incrementing by 1) is used for read (write), then the address generation is complex for the write (read) operation.

For the $2M$ filters, data from the row processors is written in consecutive locations in the MEM2 banks, but extra logic is required to generate the pattern in which the three banks are accessed [the pattern for the forward transform of (5, 3) filter can be observed in Table XI]. For the $4M$ filters, RP1 and RP2 write in consecutive locations in $MEM2_0$ and $MEM2_1$, respectively.

TABLE XII
SIZE OF MEM2 MODULE BANKS

Filter	$MEM2_0$	$MEM2_1$	$MEM2_2$	$MEM2_3$
(5,3)	1 row	1 row	1 row	1 row
(13,7)	2 rows	2 rows	1 row	3 rows
(2,6)	1 row	1 row	1 row	2 rows
(2,10)	2 rows	2 rows	1 row	4 rows
(9,7)	17 *(elements)*	17 *(elements)*	-	-
(6,10)	19 *(elements)*	19 *(elements)*	-	-

TABLE XIII
SIZES OF REGISTER FILES

Filter	REG1	REG2
(5,3)	2	1
C(13,7)	4	1
S(13,7)	4	1
(2,6)	8	5
(2,10)	10	5
(9,7)	2	2
(6,10)	2	3

TABLE XIV
TIME REQUIRED FOR ONE LEVEL OF DECOMPOSITION OF A $N \times N$ BLOCK

Filter	Total Time required
(5,3)	$2\lfloor N/2 \rfloor$+3Ta+2Ts+N+5+$\lfloor N/2 \rfloor$N
(13,7)	7N+6Ta+2Tm+4+$\lfloor N/2 \rfloor$2N
(2,6)	3$\lfloor N/2 \rfloor$+6Ta+Tm+3+$\lfloor N/2 \rfloor$N
(2,10)	5N+7Ta+Tm+ 3+$\lfloor N/2 \rfloor$2N
(9,7)	2(4Ta+6Tm+6+ $\lfloor N/2 \rfloor$N)
(6,10)	2(3Ta+6Tm+6+$\lfloor N/2 \rfloor$2N)

V. TIMING

The total time required for one level of decomposition of an $N \times N$ block for all the filters is given in Table XIV. Here, Ta is the delay of the adder, Ts is the delay of the shifter, and Tm is the delay of the multiplier. To obtain the latency for a filter, we need the start time of CP2, which depends on the number of rows CP1 has to finish before CP2 can start and the start time of CP1. The first factor would be a multiple N or $2N$, and the latter

TABLE XV
PRELIMINARY GATE COUNT ESTIMATES AND NUMBER OF COMPONENTS USED IN THE PROPOSED ARCHITECTURE

Component	Gate count	Number of units
Adder (16b+16b)	400	8
Shifter (16b, 1&2 bits ;L& R shift)	600	4
Multiplier (16bx10b)	1800	4
Register (16 bit)	100	30
Memory (2–port/ byte)	225 μ^2/byte	(129x130x2) + (129x2x4) bytes

factor would be a multiple of $\lfloor N/2 \rfloor$ or N based on whether data is generated every cycle [(5, 3), (9, 7), and (2, 6) filters] or in every alternate cycle [(13, 7) and (2, 10) filters].

For example, the latency for the (5, 3) filter is $2\lfloor N/2 \rfloor + 3Ta + 2Ts + N + 5$. Since we need $\lfloor N/2 \rfloor N$ cycles to complete one level of transform in both the dimensions on an $N \times N$ block, the time required for the (5, 3) filter is $2\lfloor N/2 \rfloor + 2Ta + 2Ts + N + 3 + \lfloor N/2 \rfloor N$.

VI. IMPLEMENTATION

We have developed a behavioral VHDL model of an architecture capable of carrying out the forward and inverse transform of (5, 3) and (9, 7) filters. The memories are simulated as arrays. The data path is 16 bits wide. The adder and shifter are assumed to have a one clock cycle delay, where as the multiplier has a four cycle delay and is pipelined to four levels. The VHDL simulations and the C code simulations match exactly. The data path units have been synthesized. The preliminary gate count (2-input NAND gate equivalents) of the data path units and number of units used in the architecture are provided in Table XV. The memory required, assuming a 129×129 block, is also provided in the table. The estimated area of the proposed architecture, assuming control is 20% of datapath area, in 0.18 μ technology is 2.8 mm square. The estimated frequency of operation is 200 MHz. The frequency is set by the time required for the dual access in a dual port memory.

VII. CONCLUSION

In this paper, we propose a VLSI architecture to implement the seven filters recommended in the upcoming JPEG2000 standard using the lifting scheme. The architecture consists of two row processors, two column processors, and two memory modules, each consisting of four banks. The processors are very simple and consist of two adders, one multiplier, and one shifter. The width of the data path is determined to be 16 bits for lossless/near lossless performance. The architecture has been designed to generate an output every cycle for the JPEG2000 part I default filters. Details of the schedule and timing performance have been included in the paper. The architecture has been implemented using behavioral VHDL. The estimated area of the proposed architecture in 0.18 μ technology is 2.8 mm square, and the estimated frequency of operation is 200 MHz.

APPENDIX

- $\mathbf{C(13,7)/S(13,7)}$

$$M_1 = \begin{bmatrix} 1 & a & 0 & b & 0 & \cdot & \cdot & \cdot & \cdot & \cdot & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & a+b & 1 & a & 0 & b & 0 & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdot & \cdot & \cdot \\ 0 & b & 0 & a & 1 & a & 0 & b & 0 & \cdot & \cdot \\ \cdot & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdot \\ \cdot & \cdot & 0 & b & 0 & a & 1 & a & 0 & b & 0 \\ \cdot & \cdot & \cdot & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & 0 & b & 0 & a & 1 & a+b & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & \cdot & \cdot & \cdot & \cdot & \cdot & 0 & b & 0 & a & 1 \end{bmatrix}$$

$$M_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdot & \cdot & \cdot & \cdot & \cdot & 0 \\ 0 & 1 & c & 0 & d & 0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & c+d & 1 & c & 0 & d & 0 & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdot & \cdot \\ \cdot & 0 & d & 0 & c & 1 & c & 0 & d & 0 & \cdot \\ \cdot & \cdot & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \cdot & \cdot & \cdot & 0 & d & 0 & c & 1 & c+d & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & 0 & d & 0 & c & 1 & 0 \\ 0 & \cdot & \cdot & \cdot & \cdot & \cdot & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

where $a = -0.5625, b = 0.0625, c = 0.3125$, and $d = -0.0625$ for the $[\mathbf{C}(\mathbf{13},\mathbf{7})]$ filter, where $a = -0.5625, b = 0.0625, c = 0.281\,25, d = -0.3125$ for the $[\mathbf{S}(\mathbf{13},\mathbf{7})]$ filter.

- $(\mathbf{2},\mathbf{6})$

$$M_1 = \begin{bmatrix} 1 & 0 & 0 & \cdot & \cdot & \cdot & \cdot & \cdot & 0 \\ 0 & 1 & 0 & 0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 1 & 0 & 0 & \cdot & \cdot & \cdot & \cdot \\ \cdot & 0 & 1 & 1 & 0 & 0 & \cdot & \cdot & \cdot \\ \cdot & \cdot & 0 & 0 & 1 & 0 & 0 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 0 & 1 & 1 & 0 & 0 & \cdot \\ \cdot & \cdot & \cdot & \cdot & 0 & 0 & 1 & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & 0 & 1 & 1 & 0 \\ 0 & \cdot & \cdot & \cdot & \cdot & \cdot & 0 & 0 & 1 \end{bmatrix}$$

$$M_2 = \begin{bmatrix} 1 & c+b & 0 & b & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & \cdot & \cdot & \cdot \\ 0 & -b & 1 & c & 0 & b & 0 & \cdot & \cdot \\ \cdot & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdot \\ \cdot & \cdot & 0 & -b & 1 & c & 0 & b & 0 \\ \cdot & \cdot & \cdot & 0 & 0 & 1 & 0 & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & 0 & -b & 1 & c & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & 0 & 0 & 1 & 0 \\ 0 & \cdot & \cdot & \cdot & \cdot & \cdot & 0 & -b & 1 \end{bmatrix}$$

where $b = -0.003\,906, c = -0.5, K_1 = 0.5, K_2 = 2$.

- For $(\mathbf{2},\mathbf{10})$, see the matrices at the bottom of the next page. where $a = 0.011\,719, b = 0.859\,38$, and $c = -0.5, K_1 = 0.5, K_2 = 2$.

- $(\mathbf{9},\mathbf{7})$

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

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

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

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

where $a=-1.586\,13, b=-0.052\,98, c=0.882\,911$, $d=-0.443\,50, K_1=1.149\,60$, and $K_2=0.869\,86$.

- $(\mathbf{6},\mathbf{10})$

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

$$M_2=\begin{bmatrix}1 & c+b & 0 & c & 0 & \cdot & \cdot & \cdot & 0\\ 0 & 1 & 0 & 0 & 0 & 0 & \cdot & \cdot & \cdot\\ 0 & 0 & 1 & b & 0 & c & 0 & \cdot & \cdot\\ \cdot & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdot\\ \cdot & \cdot & 0 & 0 & 1 & b & 0 & c & 0\\ \cdot & \cdot & \cdot & 0 & 0 & 1 & 0 & 0 & 0\\ \cdot & \cdot & \cdot & \cdot & 0 & 0 & 1 & b & 0\\ \cdot & \cdot & \cdot & \cdot & \cdot & 0 & 0 & 1 & 0\\ 0 & \cdot & \cdot & \cdot & \cdot & \cdot & 0 & 0 & 1\end{bmatrix}$$

$$M_3=\begin{bmatrix}1 & 0 & 0 & \cdot & \cdot & \cdot & \cdot & \cdot & 0\\ 0 & 1 & 0 & 0 & \cdot & \cdot & \cdot & \cdot & \cdot\\ 0 & 0 & 1 & 0 & 0 & \cdot & \cdot & \cdot & \cdot\\ 0 & 0 & e & 1 & 0 & 0 & \cdot & \cdot & \cdot\\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdot & \cdot\\ \cdot & 0 & d & 0 & e & 1 & 0 & 0 & \cdot\\ \cdot & \cdot & 0 & 0 & 0 & 0 & 1 & 0 & 0\\ \cdot & \cdot & \cdot & 0 & d & 0 & e+d & 1 & 0\\ 0 & \cdot & \cdot & \cdot & 0 & 0 & 0 & 0 & 1\end{bmatrix}$$

$$M_4=\begin{bmatrix}1 & g+f & 0 & f & 0 & \cdot & \cdot & \cdot & 0\\ 0 & 1 & 0 & 0 & 0 & 0 & \cdot & \cdot & \cdot\\ 0 & -f & 1 & g & 0 & f & 0 & \cdot & \cdot\\ \cdot & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdot\\ \cdot & \cdot & 0 & -f & 1 & g & 0 & f & 0\\ \cdot & \cdot & \cdot & 0 & 0 & 1 & 0 & 0 & 0\\ \cdot & \cdot & \cdot & \cdot & 0 & -f & 1 & g & 0\\ \cdot & \cdot & \cdot & \cdot & \cdot & 0 & 0 & 1 & 0\\ 0 & \cdot & \cdot & \cdot & \cdot & \cdot & 0 & -f & 1\end{bmatrix}$$

where $a=-0.369\,536, b=-0.427\,80, c=-0.119\,532, d=-0.090\,075, e=0.872\,739, g=-0.572\,909, f=0.224\,338, K_1=0.874\,919$, and $K_2=1.142\,963$.

$$M_1=\begin{bmatrix}1 & 0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 0\\ 0 & 1 & 0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ 0 & 0 & 1 & 0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & 0 & 1 & 1 & 0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & 0 & 0 & 1 & 0 & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & 0 & 1 & 1 & 0 & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & 0 & 0 & 1 & 0 & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & 0 & 1 & 1 & 0 & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 0 & 0 & 1 & 0 & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 0 & 1 & 1 & 0\\ 0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 0 & 0 & 1\end{bmatrix},\quad M_2=\begin{bmatrix}1 & c+b & 0 & b-a & 0 & -a & 0 & \cdot & \cdot & \cdot & 0\\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdot & \cdot & \cdot\\ 0 & -(b+a) & 1 & c & 0 & b & 0 & -a & 0 & \cdot & \cdot\\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdot\\ 0 & a & 0 & -b & 1 & c & 0 & b & 0 & -a & 0\\ \cdot & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0\\ \cdot & \cdot & 0 & a & 0 & -b & 1 & c & 0 & b & 0\\ \cdot & \cdot & \cdot & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\ \cdot & \cdot & \cdot & \cdot & 0 & a & 0 & -b & 1 & c & 0\\ \cdot & \cdot & \cdot & \cdot & \cdot & 0 & 0 & 0 & 0 & 1 & 0\\ 0 & \cdot & \cdot & \cdot & \cdot & \cdot & 0 & 0 & 0 & a-b & 1\end{bmatrix}$$

REFERENCES

[1] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting schemes," *J. Fourier Anal. Appl.*, vol. 4, pp. 247–269, 1998.

[2] W. Sweldens, "The lifting scheme: A new philosophy in biorthogonal wavelet constructions," in *Proc. SPIE*, vol. 2569, 1995, pp. 68–79.

[3] JPEG2000 Committee Drafts [Online]. Available: http://www.jpeg.org/CDs15444.htm

[4] *JPEG2000 Verification Model 8.5 (Technical Description)*, Sept. 13, 2000.

[5] K. Andra, C. Chakrabarti, and T. Acharya, "A VLSI architecture for lifting based wavelet transform," in *Proc. IEEE Workshop Signal Process. Syst.*, Oct. 2000, pp. 70–79.

[6] M. Vishwanath, R. Owens, and M. J. Irwin, "VLSI architectures for the discrete wavelet transform," *IEEE Trans. Circuits Syst. II*, vol. 42, pp. 305–316, May 1995.

[7] J. S. Fridman and E. S. Manolakos, "Discrete wavelet transform: Data dependence analysis and synthesis of distributed memory and control array architectures," *IEEE Trans. Signal Processing*, vol. 45, pp. 1291–1308, May 1997.

[8] T. Acharya, "A high speed systolic architecture for discrete wavelet transforms," in *Proc. IEEE Global Telecommun. Conf.*, vol. 2, 1997, pp. 669–673.

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

[10] 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, June 1996.

[11] C. Chakrabarti and M. Vishwanath, "Efficient realizations of the discrete and continuous wavelet transforms: From single chip implementations to mappings on SIMD array computers," *IEEE Trans. Signal Processing*, vol. 43, pp. 759–771, Mar. 1995.

[12] W. Jiang and A. Ortega, "Lifting factorization-based discrete wavelet transform architecture design," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, pp. 651–657, May 2001.

[13] C. Diou, L. Torres, and M. Robert, "A wavelet core for video processing," presented at the IEEE Int. Conf. Image Process., Sept. 2000.

[14] G. Lafruit, L. Nachtergaele, J. Bormans, M. Engels, and I. Bolsens, "Optimal memory organization for scalable texture codecs in MPEG-4," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, pp. 218–243, Mar. 1999.

[15] M. Ferretti and D. Rizzo, "A parallel architecture for the 2-D discrete wavelet transform with integer lifting scheme," *J. VLSI Signal Processing*, vol. 28, pp. 165–185, July 2001.

[16] A. R. Calderbank, I. Daubechies, W. Sweldens, and B.-L. Yeo, "Wavelet transforms that map integers to integers," *Appl. Comput. Harmon. Anal.*, vol. 5, pp. 332–369, July 1998.

[17] USC-SIPI Image Database [Online]. Available: http://sipi.usc.edu/services/database/Database.html

[18] M. D. Adams and F. Kossentini, "Reversible integer-to-integer wavelet transforms for image compression: Performance evaluation and analysis," *IEEE Trans. Image Processing*, vol. 9, pp. 1010–1024, June 2000.

Kishore Andra received the B.Tech. degree in electrical and electronics engineering from the J.N.T. University, Anantapur, India, in 1994, the M.S. degree from the Indian Institute of Technology, Madras, and the Ph.D. degree from Arizona State University, Tempe, both in electrical engineering, in 1997 and 2001, respectively. As part of his Ph.D. thesis, he developed an architecture for the JPEG2000 still image compression standard.

Currently, he is with Maxim Integrated Products, Sunnyvale, CA, working on the design of low-power and high-performance mixed signal intergrated circuits.

Chaitali Chakrabarti (M'90) received the B.Tech. degree in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur, in 1984 and the M.S. and Ph.D. degrees in electrical engineering from the University of Maryland, in 1986 and 1990, respectively.

Since August 1990, she has been with the Department of Electrical Engineering, Arizona State University (ASU), Tempe, where she is currently an Associate Professor. Her research interests are in the areas of low-power systems design including memory optimization, high-level synthesis and compilation, and VLSI architectures and algorithms for signal processing, image processing, and communications. She is an Associate Ediotr for *the Journal of VLSI Signal Processing Systems*.

Dr. Chakrabarti is a member of the Center of Low Power Electronics (jointly funded by the National Science Foundation, the state of Arizona, and the member companies) and the Telecommunications Research Center. 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.

Tinku Acharya (SM'01) received the B.Sc. (Honors) degree in physics and B.Tech. and M.Tech. degrees in computer science from teh University of Calcutta, Calcutta, India, in 1983, 1987, and 1989, respectively. He received the Ph.D. degree in computer science from the University of Central Florida, Orlando, in 1994.

Currently, he is a Principal Engineer with the Intel Architecture Group, Intel Corporation, Tempe, AZ, and an Adjunct Professor with the Department of Electrical Engineering, Arizona State University, Tempe. Before joining Intel Corporation in 1996, he was a Consulting Engineer with AT&T Bell Laboratories from 1995 to 1996, was a Faculty Member at the Institute of Systems Research, University of Maryland, College Park, from 1994 to 1995, and held Visiting Faculty positions at Indian Institute of Technology (IIT), Kharagpur (on several occassions from 1998 to 2001). He has contributed to more than 50 technical papers published in international journals, conferences, and book chapters. He holds 27 U.S. patents, and more than 80 patents are pending. His current interest of research includes VLSI architectures and algorithms, electronic and digital image processing, data/image/video compression, and media processing algorithms in general.

Dr. Acharya serves on the U.S. National Body of the JPEG2000 committee.