A VLSI ARCHITECTURE FOR LIFTING-BASED WAVELET TRANSFORM

Kishore Andra
Electrical Engg. Dept.
Arizona State University
Tempe, Arizona

Chaitali Chakrabarti
Electrical Engg. Dept.
Arizona State University
Tempe, Arizona

Tinku Acharya
INTEL Corporation,
Chandler, Arizona

Abstract : The Discrete Wavelet Transform (DWT) is the basis for many image compression techniques, such as the upcoming JEPG2000. Lifting-based DWT requires fewer computations compared to the traditional convolution-based approach. In this paper, we propose a VLSI architecture to compute lifting-based 2D DWT, for a set of seven filters recommended in the JPEG2000 verification model. The architecture produces transform coefficients in every clock cycle for three of the filters and in every alternate cycle for the rest of the filters. We also present an efficient memory organization to address the high memory bandwidth requirements. The performance metrics of the proposed architecture have also been furnished.

1. 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. One drawback of the conventional convolution based DWT is that it requires a lot of computations. The lifting based scheme for the DWT [1],[2] helps in overcoming this drawback and has been chosen in the upcoming JPEG2000 [3] standard.

The main feature of the lifting based DWT scheme is to break up the high pass and low pass filters into a sequence of upper and lower triangular matrices, and convert the filter implementation into banded matrix multiplications. The factorization of the filters is not unique [2]. In this paper, we have chosen factorizations, based on symmetry and fewer number of computations, making it suitable for a low-power VLSI implementation. Lifting has other advantages, such as "in-place" computation of the DWT, integer-to-integer wavelet transforms (which are useful for lossless coding) etc. In JPEG2000 Verification Model[3], the following wavelet filters (5,3), (9,7), C(13,7), S(13,7), (2,6), (2,10) and (6,10) have been proposed. In this paper we propose a unified architecture capable of executing all these filters using the lifting scheme.

The proposed architecture computes multi-level DWT, 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 fine for filters that require two banded-matrix multiplications, filters that require four banded-matrix multiplications require all four processors to compute along the rows or along the columns. Since different filters have different computational requirements, we focus on the configuration that ensures an output in every cycle for the chosen filters [(5,3) and (9,7)]. The outputs generated by the row and column processors (that are used for further computations) are stored in memory

0-7803-6488-0/00/$10.00 © 2000 IEEE

modules. The memory modules are divided into multiple banks and an innovative data arrangement has been employed to accommodate high computational bandwidth requirements.

The rest of the paper is organized as follows. In section 2, we give a brief overview of the lifting scheme. The proposed architecture and its memory organization are presented in section 3. We present the timing performance of the architecture in section 4. We conclude the paper in section 5.

2. LIFTING BASED DISCRETE WAVELET TRANSFORM

The basic principle of the lifting scheme [1],[2] is to factorize the polyphase matrix of a wavelet filter into a sequence of alternating upper and lower triangular matrices and a diagonal matrix with constants. The factorization is obtained by using an extension of the Euclidean algorithm. The resulting formulation can be implemented by means of banded matrix multiplications.

Let $\tilde{h}(z)$ and $\tilde{g}(z)$ be the low pass and high pass analysis filters and h(z) and g(z) be the low pass and high pass synthesis filters. The filters can be divided into even and odd parts as

$$\tilde{h}(z)=\tilde{h}_e(z^2)+z^{-1}\tilde{h}_o(z^2) \qquad h(z)=h_e(z^2)+z^{-1}h_o(z^2)$$
$$\tilde{g}(z)=\tilde{g}_e(z^2)+z^{-1}\tilde{g}_o(z^2) \qquad g(z)=g_e(z^2)+z^{-1}g_o(z^2)$$

The polyphase matrices are then defined as

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

It has been shown in [2] that if h,g is a complementary filter pair, then P(z) can always be factored into lifting steps as shown below :

$$P_1(z)=\begin{bmatrix}K_1 & 0\\ 0 & K_2\end{bmatrix}\prod_{i=1}^{m}\begin{bmatrix}1 & s_i(z)\\ 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\ t_i(z) & 1\end{bmatrix} \text{ or } P_2(z)=\begin{bmatrix}K_1 & 0\\ 0 & K_2\end{bmatrix}\prod_{i=1}^{m}\begin{bmatrix}1 & 0\\ t_i(z) & 1\end{bmatrix}\begin{bmatrix}1 & s_i(z)\\ 0 & 1\end{bmatrix}$$

where K_1 and K_2 are constants. The two types of lifting schemes are shown in Fig.1.

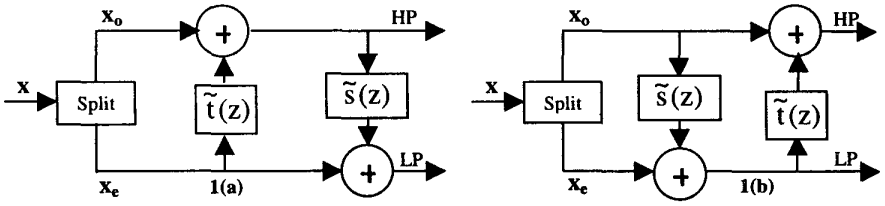

Fig.1 Lifting Schemes. 1(a) Scheme 1, 1(b) Scheme 2

In Scheme 1 (Fig. 1(a)), which corresponds to the $P_1(z)$ factorization, the low-pass samples (even terms) are multiplied by the time domain equivalent of $\tilde{t}(z)$, and are added to the high pass samples (odd terms) in the first step. In the second step, the updated high pass samples are multiplied by the time domain equivalent of $\tilde{s}(z)$ and are added to the low-pass samples. In Scheme 2 (Fig. 1(b)), which corresponds to the $P_2(z)$ factorization, the low-pass terms are calculated in the first step ($\tilde{s}(z)$ is used) and the high pass terms are calculated in second step ($\tilde{t}(z)$ is used).

Example : Let us consider the (5,3) filter, with the following filter coefficients :
Low-pass ($\tilde{h}(z)$) : (6/8, 2/8, -1/8) & High-pass ($\tilde{g}(z)$) : (1, -1/2)
The polyphase matrix of the above filter is

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

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

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

If the signal is numbered from 0, and if even terms are considered the low pass values and the odd terms the high pass values, we can interpret the above matrices in the time domain as –

$$y_{2i+1} = -0.5(x_{2i}+x_{2i+2}) + x_{2i+1} \qquad y_{2i} = 0.25(y_{2i+1}+y_{2i+3}) + x_{2i}$$

where, x's are the signal values and y's are the transformed signal values. The corresponding matrices M_1 and M_2 are shown below (where a = -0.5 and b = 0.25). The transform of the signal X is then obtained as $X*M_1*M_2$.

$$M_1=\begin{bmatrix} 1 & a & 0 & . & . & . & . & . & 0 \\ 0 & 1 & 0 & 0 & . & . & . & . & . \\ 0 & a & 1 & a & 0 & . & . & . & . \\ . & 0 & 0 & 1 & 0 & 0 & . & . & . \\ . & . & 0 & a & 1 & a & 0 & . & . \\ . & . & . & 0 & 0 & 1 & 0 & 0 & . \\ . & . & . & . & 0 & a & 1 & a & 0 \\ . & . & . & . & . & 0 & 0 & 1 & 0 \\ 0 & . & . & . & . & . & 0 & a & 1 \end{bmatrix}, M_2 = \begin{bmatrix} 1 & 0 & 0 & . & . & . & . & . & 0 \\ 0 & 1 & b & 0 & . & . & . & . & . \\ 0 & 0 & 1 & 0 & 0 & . & . & . & . \\ . & 0 & b & 1 & b & 0 & . & . & . \\ . & . & 0 & 0 & 1 & 0 & 0 & . & . \\ . & . & . & 0 & b & 1 & b & 0 & . \\ . & . & . & . & 0 & 0 & 1 & 0 & 0 \\ . & . & . & . & . & 0 & b & 1 & 0 \\ 0 & . & . & . & . & . & 0 & 0 & 1 \end{bmatrix}$$

In this work we have considered a block wavelet transform with a single sample overlap as recommended in JPEG2000. As a result, the length (row and column) is odd and also, the first and last values in the input signal do not change on applying the transform. The wavelet filters (5,3), C(13,7), S(13,7), (2,6), (2,10), are factorized into 2 matrices M_1 and M_2, while filters (9,7) and (6,10) are factorized into 4 matrices M_1,M_2,M_3,M_4. Filters (2,6), (2,10), (9,7) and (6,10) require an additional diagonal matrix with constant coefficients. The width of the band of the matrices for the various filters is given in Table 1. The wider the band, more the number of computations and more the amount of storage for the intermediate results.

We classify the filters based on the number of factorization matrices: a two matrix factorization is denoted by 2M and a four matrix factorization by 4M. Furthermore, filters (5,3), C(13,7), S(13,7) and (9,7), use lifting scheme 1 (Fig.1(a)), while (2,6), (2,10) and (6,10) use lifting scheme 2 (Fig.1(b)). The number of computations required for calculation of a hp, lp pair using convolution and lifting scheme are given in Table 2.

Filter	M1	M2	M3	M4	Diag
(5,3)	3	3	-	-	-
C(13,7)	7	7	-	-	-
S(13,7)	7	7	-	-	-
(2,6)	2	5	-	-	✔
(2,10)	2	9	-	-	✔
(9,7)	3	3	3	3	✔
(6,10)	2	4	4	5	✔

Table 1 Widths of the bands in the matrices

The reduction in the number of multiplications for the lifting scheme is significant for odd tap filters compared to convolution. For even tap filters, the convolution scheme has fewer or 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 high throughput and low power applications.

	For an lp and hp pair			
Filter	Multiplications		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

Table 2 Computational complexity comparison

3. PROPOSED VLSI ARCHITECTURE

The proposed architecture calculates DWT in the row-column fashion on a block of data. The architecture reads in a block of data of size NxN, carries out the DWT, and outputs the LH, HL, HH data at each level of decomposition. The LL data is used for the next level of decomposition. The block diagram of the architecture is given in Fig.2.

The architecture consists of two Row Processors (RP1,RP2), two Column Processors (CP1,CP2), two memory modules (MEM1, MEM2), two register files (REG1,REG2) and four shifter and multiplier modules(Sh/Mult). Processors RP1 and RP2 read the data from MEM1. They 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*, writes the data into MEM2 and Ext.MEM. CP2, reads the data from MEM2, performs the column wise DWT along the *rows* that CP1 did not work on, and writes the data to MEM1/external memory based on which band the transform element belongs to.

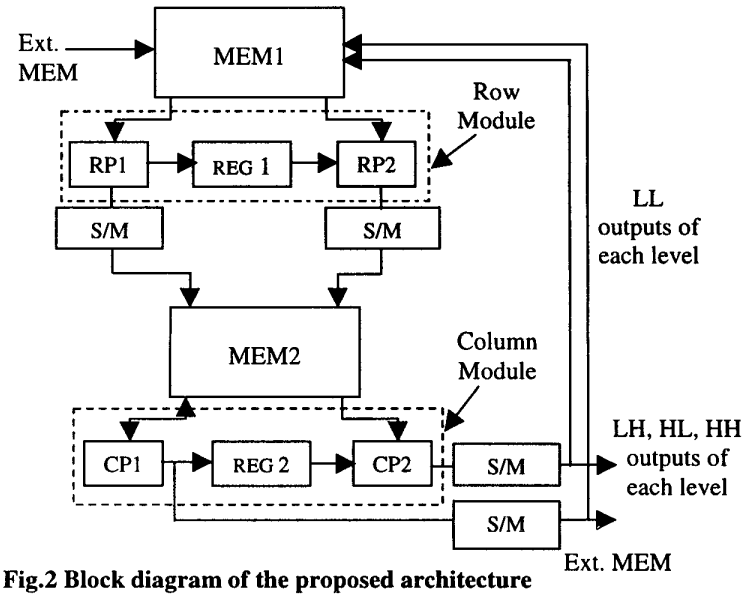

Fig.2 Block diagram of the proposed architecture

The data flow given above holds good for the 2M case (i.e. factorization into two matrices). For the 4M case, there are two passes. In each of the passes RP1 and RP2 read in the data, execute the first two matrix multiplications and write the result into MEM2. CP1 and CP2 execute the next two matrix multiplications and write the result into MEM1.

In the 2M case, the latency would be very large if the column processors have to wait for all the rows to finish before they can start. To overcome this, the column processors also have to work row-wise. This is illustrated in Fig.3 for the (5,3) filter for a signal of length 5.

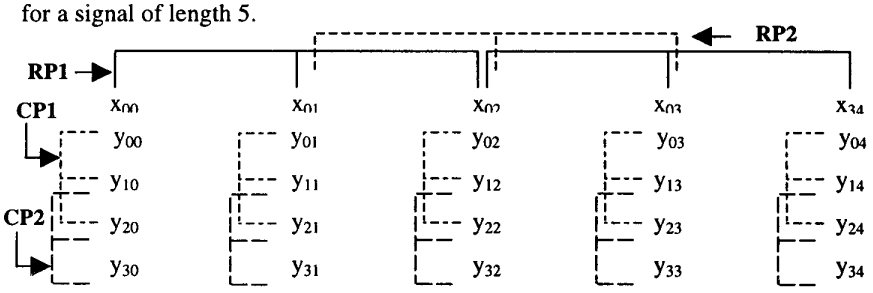

Fig.3 Illustration of row and column processor data access patterns

RP1 calculates the odd valued (high-pass) samples ($x_{k,2i}$, $x_{k,2i+2}$, $x_{k,2i+1} \rightarrow y_{k,2i+1}$) and RP2 calculate the even valued (low-pass) ($y_{k,2i+1}$, $y_{k,2i+2}$, $x_{k,2i} \rightarrow y_{k,2i}$) of all the rows. So each row processor calculates $\lfloor N/2 \rfloor$ samples of N rows. CP1 calculates *all* the samples ($y_{2i,j}$, $y_{2i+2,j}$, $y_{2i+1,j} \rightarrow z_{2i+1,j}$), $0 \le j \le N$, along the columns for *odd* numbered rows while, CP2 calculates *all* the samples ($z_{2i+1,j}$, $z_{2i+3,j}$, $y_{2i+3,j} \rightarrow z_{2i+2,j}$) along the

columns for *even* numbered rows. So the column processors calculate N samples along $\lfloor N/2 \rfloor$ rows.

For the proposed style of computation, where both the row and column processors work along the rows, the rows should not be calculated one after the other in sequential fashion. For example, in the (5,3) filter, we need the 0^{th} and the 2^{nd} elements to calculate the 1^{st} element. This implies that we need to finish the 0^{th} row and the 2^{nd} row before we go onto the 1^{st} row (so that CP1 can start the 1^{st} row). Table 3 describes the order in which the row processors and the column processors access the rows for a few of the filters, for a 9x9 block. Note that each filter needs a different order in which the row computations need to be finished. This order is determined by the factorization matrices. For instance, for the (5,3) filter, the row processors calculates rows in the order 0,2,1,4,3,6,5,8,7. CP1 starts as soon as the first output from row 1 is available, it calculates along row 1, then along row 3 etc.. CP2 starts after the first output from row 3 is available from CP1 and then works on rows 2,4,6. It outputs the elements of the 0^{th} and 8^{th} rows without any computation.

(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	-	-	3	-	-	1	-	*0*
1	1	*0*	4	-	-	2	2	-	3	-	-
4	-	-	1	1	-	1	-	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*						

Table 3 Row order followed by row and column processors for a 9x9 block

3.1 Row and Column Processor Design

Each filter requires a different configuration of adders and multipliers in order to generate an output every cycle. The (5,3) filter requires two adders and a shifter, and have the smallest configuration for both row and column processors. The (13,7) filter has the largest configuration (4 adders and 2 multipliers) for RP1 & CP1, and filter (2,10) has the largest configuration (5 adders, 2 multipliers and 1 shifter) for RP2 & CP2. The total hardware required for all the four processors for different filters, to generate an output every clock cycle is given in Table 4.

From Table 4 it can be seen that we need 18 adders, 8 multipliers and 4 shifters in order for every filter to generate an output each clock cycle. These resources are grossly underutilized for most of the filters. This prompted us to look at a configuration with fewer resources that would generate an output every clock cycle for the default JPEG2000 filters, namely the (5,3) and (9,7) filters.

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	-

Table 4 Hardware required to generate an output each clock cycle

The proposed architecture consists of four processors, where each processor consists of 2 adders, 1 multiplier and 1 shifter as shown in Fig.4. Since fewer resources are being used (8 adders, 4 multiplier and 4 shifters), an output is generated every other cycle for the (13,7), (2,10) and (6,10) filters. An output is generated every cycle for the remaining filters. Note that the MUXs at the inputs of the multipliers and adders have not been shown in Fig.4.

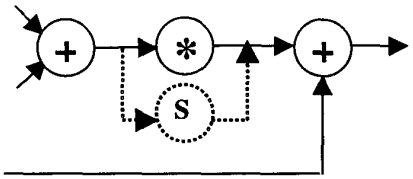

Fig.4 Basic Architecture of the each processor

3.2 Register Files

We need register files between the processors to minimize the number of memory accesses. The outputs from RP1 (CP1) are stored in REG1 (REG2) and are used by RP2 (CP2). For (2,6) and (2,10) filters the shift operation required is carried out in the RP1 (CP1) in order to keep the throughput high. Because of this, the partial sum has to be held for a time proportional to the multiplier delay. Further, we need registers (REG) at the input of RP1 and CP1 to hold values previously read from MEM1. Table 5 lists the number of registers required.

Filter	REG1	REG2	REG
(5,3)	2	1	2 (RP1)
(13,7)	4	1	4 (RP1)
(2,6)	3 + (Tm+1)	Tm+1	-
(2,10)	5 + (Tm+1)	Tm+1	-
(9,7)	2	2	2 (RP1)+2
(6,10)	2	3	-

Table 5 Sizes of register files

3.3 Shift and Multiply Units

These units are used to carry out the diagonal matrix part of the factorization. They are needed to multiply the low-pass values with K_1 and high-pass values with K_2. In the case of (2,6) and (2,10) filters, the constants are 0.5 and 2, and so a shifter is required.

3.4 Memory Organization

In deriving the memory organization, we have assumed a maximum of 4 accesses (read + writes)/cycle/bank are possible.

MEM1: MEM1 can be written from external memory, CP1 and CP2, and read by RP1 and RP2. MEM1 is broken up into two banks $MEM1_0$, $MEM1_1$. While $MEM1_0$ contains the even samples, $MEM1_1$ contains the odd samples. We split the memory into two banks, since in each cycle, the row processors need a minimum of three reads and the column processors require two writes. So a minimum of 5 accesses are needed per cycle, which a single memory module cannot support.

MEM2: MEM2 is broken up into 4 banks ($MEM2_0$, $MEM2_1$, $MEM2_2$, $MEM2_3$). RP1 and RP2 write into $MEM2_0$ $MEM2_1$ & $MEM2_2$ and they are read by CP1. CP1 writes into $MEM2_3$ and it is read by CP2. $MEM2_0$ contains the odd elements, while $MEM2_1$ contains the even elements of all the rows stored for the (5,3) and (13,7) filters. For (2,6), (2,10) filters, $MEM2_0$ contains the even elements, while $MEM2_1$ contains the odd elements. The 0^{th} and N^{th} values of all the rows are in $MEM2_2$ (this is required to maintain the number of memory access at 4). $MEM2_3$ contains the odd rows (generated by CP1) for (5,3), (13,7) and even rows for (2,6), (2,10) filters. Filters (9,7) and (6,10) use only $MEM2_0$, $MEM2_1$ and the data contained there is still the same as in the 2M case. It should be noted that amount of time data is available in the banks depends upon the filters.

3.4.1 Memory Sizes

MEM1: To support the (9,7) and (6,10) filters, we need a MEM1 of size NxN, where N is the maximum allowed row/column size. The size of $MEM1_0$ is $(\lfloor N/2 \rfloor+1)xN$, while $MEM1_1$ is $\lfloor N/2 \rfloor xN$.

MEM2: The number of rows required in $MEM2_0$ is fixed by the following consideration. CP1 starts off as soon as the first odd row output is available from RP1 and works on all the N elements in the row. In the same time, RP1 works on an even row and the next odd row (Table 3), but on $\lfloor N/2 \rfloor$ elements in each row. Due to this size difference, after every row that CP1 finishes, it lags behind RP1 by one element. So after computing a $\lfloor N/2 \rfloor$ rows CP1 lags behind by $\lfloor N/2 \rfloor$ samples. Thus we need an extra row of memory to compensate for this mismatch. This is in addition to the row that is used to hold the RP1 output.

The number of rows in $MEM2_1$ can be deduced by studying the order in which the computations are done (see Table 3). For example, for the (5,3) filter, we observe that before row 2 computations can be completed (by CP2), row 4 and row 6 are already generated (by RP2). Thus, we need 3 rows to hold the even row outputs. $MEM2_2$ has same number of rows as the number of even rows, with two elements in each row.

Filter	$MEM2_0$	$MEM2_1$	$MEM2_2$	$MEM2_3$
(5,3)	$2\lfloor N/2 \rfloor$	$3(\lfloor N/2 \rfloor-1)$	3*2	$2\lfloor N/2 \rfloor$
(13,7)	$2\lfloor N/2 \rfloor$	$4(\lfloor N/2 \rfloor-1)$	4*2	$4\lfloor N/2 \rfloor$
(2,6)	$2(\lfloor N/2 \rfloor-1)$	$3\lfloor N/2 \rfloor$	2*2	$3\lceil N/2 \rceil$
(2,10)	$2(\lfloor N/2 \rfloor-1)$	$4\lfloor N/2 \rfloor$	2*2	$5\lceil N/2 \rceil$
(9,7)	2Ta+3Tm+3 *(CP1-RP1)*		-	-
(6,10)	2Ta+3Tm+5 *(CP2-RP2)*		-	-

Table 7 Size of memory required (N is the number of elements per row).

The size of $MEM2_3$ can be deduced from Table 3. For example, to calculate along row 2, CP2 requires 2 rows : row 1 (generated by CP1) and row 3 (being generated by CP1). The size of the memory required for the 4M case is calculated by taking the maximum of the differences of latencies of RP1 & CP1 and RP2 & CP2. The maximum number of elements for all the filters is given in Table 7.

3.5 Schedule

We have generated a detailed schedule for each of the filters by hand. We provide a snap shot of the schedule for the (5,3) filter applied on a 9x9 block in Table 6. It is assumed that the delay of the adder and shifter are equal to 1.

Time	RP1			RP2		
	Adder 1	Shifter	Adder 2	Adder 1	Shifter	Adder 2
1	$x_{0,0},x_{0,2}$	-	-	-	-	-
2	$x_{0,2},x_{0,4}$	RA1	-	-	-	-
3	$x_{0,4},x_{0,6}$	RA1	RS-$x_{0,1}$ = $y_{0,1}$	-	-	-
4	$x_{0,6},x_{0,8}$	RA1	RS-$x_{0,3}$ = $y_{0,3}$	-	-	-
5	$x_{2,0},x_{2,2}$	RA1	RS-$x_{0,5}$ = $y_{0,5}$	$y_{0,1},y_{0,3}$	-	-
6	$x_{2,2},x_{2,4}$	RA1	RS-$x_{0,7}$ = $y_{0,7}$	$y_{0,3},y_{0,5}$	RA1	-
7	$x_{2,4},x_{2,6}$	RA1	RS-$x_{2,1}$ = $y_{2,1}$	$y_{0,5},y_{0,7}$	RA1	RS+$x_{0,2}$ = $y_{0,2}$
8	$x_{2,6},x_{2,8}$	RA1	RS-$x_{2,3}$ = $y_{2,3}$	-	RA1	RS+$x_{0,4}$ = $y_{0,4}$
9	$x_{1,0},x_{1,2}$	RA1	RS-$x_{2,5}$ = $y_{2,5}$	$y_{2,1},y_{2,3}$	RA1	RS+$x_{0,6}$ = $y_{0,6}$
10	$x_{1,2},x_{1,4}$	RA1	RS-$x_{2,7}$ = $y_{2,7}$	$y_{2,3},y_{2,5}$	RA1	-
11	$x_{1,4},x_{1,6}$	RA1	RS-$x_{1,1}$ = $y_{1,1}$	$Y_{2,5},y_{2,7}$	RA1	RS+$x_{2,2}$ = $y_{2,2}$

Table 6. A part of the schedule for RP1 and RP2 for the (5,3) filter

The above schedule should be read as follows. In the 5th cycle, the elements in Adder 1 column ($x_{2,0},x_{2,2}$) are added and stored in the register at its output. The shifter (Shifter column) reads this sum in the next cycle (6th cycle) and carries out required number of shifts (one in this case as a = -0.5) and stores the data in the register at its output. The second adder (Adder 2) reads the shifted value of the sum and subtracts it from a element supplied by the memory ($x_{2,1}$) to generate $y_{2,1}$ in the next cycle (7th cycle). The output of the second adder is stored in a suitable memory location. The Time column represent the number of sample periods from the start of the computation.

4. TIMING

The total time required (for both maximum hardware case and for the architecture described in this paper), for one level of decomposition of an NxN block, for all the filters is given in Table 8. Here, Ta is the delay of the adder/shifter, and Tm is the delay of the multiplier. To obtain the total time, we need (i) the start time of RP2 which depends on the latency of RP1 (ii) the start time of CP1, which depends on the number of rows RP1 has to calculate before the first odd/even row is calculated, and thus is a multiple of $\lfloor N/2 \rfloor$ for the filters where the data is generated every cycle i.e. (5,3), (9,7) and (2,6) or a multiple of N for filters where data is generated every alternate cycle i.e. (13,7) and (2,10) and (iii) the start time of CP2 which depends on

the number of rows CP1 has to finish before CP2 can start, and so it is a multiple N or 2N. For example, consider the (5,3) filter. The start time of RP2 is Ta+Tm+3, for CP1 it is $2\lfloor N/2 \rfloor$+2 and for CP2 it is $2\lfloor N/2 \rfloor$+Ta+Tm+N+3. The latency of CP2 is Ta+Tm. So the latency from the beginning of the computation would be $2\lfloor N/2 \rfloor$+2Ta+2Tm+N+3. Since we need $\lfloor N/2 \rfloor N$ cycles to complete one level of transform in both the dimensions on an NxN block, the time required is $2\lfloor N/2 \rfloor$+2Ta+2Tm+N+3+$\lfloor N/2 \rfloor N$.

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

Table 9 Time required for one level of decomposition of an NxN block

5. CONCLUSIONS

A unified VLSI architecture for lifting based multi-level DWT is proposed. The architecture consists of two row processors, two column processors, two memory modules and two register files. Each processor consists of 2 adders, 1 multiplier, and 1 shifter. This architecture yields an output sample in every clock cycle for default wavelet filters recommended in JPEG2000 VM and is programmable by a controller to support different hardware configurations required by different filters. We avoid a detailed discussion of the controller here.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Wim Sweldens for helping us get started on Lifting. We also thank Mr. Clemens Valens for providing the software that generates the lifting coefficients from filter coefficients.

REFERENCES

1. I. Daubechies and W. Sweldens, " Factoring wavelet transforms into lifting schemes", The J. of Fourier Analysis and Applications, Vol. 4, 247-269, 1998.
2. W. Sweldens, "The lifting scheme: A new philosophy in biorthogonal wavelet constructions", Proceedings of SPIE, 2569, 68-79, 1995.
3. JPEG2000 Verification Model 6.0.