Transactions Letters

An Efficient VLSI Implementation of the Discrete Wavelet Transform Using Embedded Instruction Codes for Symmetric Filters

Bing-Fei Wu and Yi-Qiang Hu

***Abstract*—This work presents a VLSI design rule, namely, an embedded instruction code (EIC), for the discrete wavelet transform (DWT). Our approach derives from the essential computations of DWT, and we establish a set of multiplication instructions, MUL, and the addition instruction, ADD. In addition, we propose a parallel arithmetic logic unit (PALU) with two multipliers and four adders, called 2M4A. With these requirements, the DWT computation paths can be calculated more efficiently with limited PALUs. Furthermore, since the EIC is operated under the PALU, the number of needed inner registers depends on the wavelet filters' length. Besides, the boundary problem of DWT has also been resolved by the symmetric extension. Moreover, the two-dimensional inverse DWT (2-D IDWT) can be completed using the same PALU for 2-D DWT; the only changes needed to be made are the instruction codes and coefficients. Our chip supports up to six levels of decomposition and versatile image specifications, e.g., VGA, MPEG-1, MPEG-2 and 1024 $\times$ 1024 image sizes.**

***Index Terms*—Two-dimensional discrete wavelet transform, VLSI.**

I. Introduction

LATELY, the hardware development of discrete wavelet transform (DWT) has made rapid advance [1]–[4]. Its main applications include signal, image and video processing; and it specifically puts highlights on data compression. Recently, the standard protocol of still images, JPEG 2000 [5], has employed the DWT for transform coding. In this paper, we design the integrated hardware containing the functions of an one-dimensional DWT (1-D DWT) and inverse DWT (1-D IDWT) [6]. Thus, we ultimately synthesize the architecture with the functions of a two-dimensional DWT (2-D DWT) and IDWT (2-D IDWT).

Knowles's DWT [7] adopted a number of multiplexers, which is not suitable for hardware implementation. Hence, Vishwanath *et al.* proposed an innovative design in [2] and [4], which is called recursive pyramid algorithm (RPA), and their the implementations are formed in a systolic array. It rids the drawback of Mallat's pyramid algorithm (PA) [8], whose implementations need the feedback storages. Lately, the utilizations of systolic in DWT have been thoroughly studied in researches [3], [9]–[13]. To achieve better computation efficiency, Shunjun *et al.* [10] and Marino *et al.* [12] used the parallel and pipeline techniques, respectively. Moreover, the border problem [3] should be careful since the perfect reconstruction (PR) [1], [6], [19] is very critical in DWT. Ferretti *et al.* [3] proposed the modified RPA to solve this problem.

The realizations of 2-D DWT have two difference approaches: separable and nonseparable. In [20]–[22], the architectures were built based on the RPA and separable approach. To reduce the usage of multipliers and storages, Lee *et al.* [21] proposed a revised RPA, called, circular parallel architecture. According to experimental results, it had performed well in various wavelet filters' length and image sizes. On the other hand, to ensure flexibility in the selection of the wavelet filters' length and the decomposition level, Chen *et al.* [22] built 256 process elements to achieve the work. In addition, Ferretti *et al.* [23] analyzed the data dependencies and designed an architecture with the computer cells to achieve the integer lifting DWT. Hence, their work can reduce the recourse less than the needed of RPA. Besides, Lafruit *et al.* [24] reorganized the arithmetic data; thus, they can minimize the size and accesses of memory.

In the RPA, the number of process element contained in systolic array depends on the chosen wavelet filers. That is, if the filters' length increases, the number rises, as well. Therefore, to restrict number of process element, we present a plastic design rule, named, embedded instruction code (EIC) and propose a VLSI architecture that is organized with a parallel arithmetic logic unit (PALU). In addition, the number of multipliers and adders in PALU remains the same regardless of wavelet filters' length.

The primary concept of EIC employs the simply built-in instruction, to command the PALU for the 1-D DWT/IDWT processing. While the 1-D architecture is made up of EIC, we put the 2-D DWT into practice with separable approach. At the outset, because the essential computations of DWT/IDWT are the multiplication and addition, we can build the multiplication instruction, MUL, the addition instruction, ADD and the general-purpose registers (GPRs), to complete the work. The instructions that can be drawn up regularly would be embedded in a built-in RAM or ROM and performed by a look-up-table. With the requirements, the instructions command the PALU to enforce specific jobs, including the fetching of the filter's coefficients, multiplication, addition and the appointment of specific registers for accumulators. Furthermore, the highlight in our design is

Manuscript received January 15, 2002; revised February 13, 2003. This work was supported by 91X104-EX-91-E-FA06-4-4. This paper was recommended by Associate Editor R. Chandramouli.

The authors are with the Department of Electrical and Control Engineering, National Chiao Tung University, Hsinchu, Taiwan, 30050, R.O.C. (e-mail: bwu@cssp.cn.nctu.edu.tw; benson@cssp.cn.nctu.edu.tw).

Digital Object Identifier 10.1109/TCSVT.2003.816509

that the architecture needs only two multipliers and four adders (2M4A). We utilize 2M4A to construct PALU organization with parallel technique. Therefore, we have successfully limited hardware resource to execute accelerative the 1-D DWT/IDWT computations.

The similarity between the computations of DWT and IDWT has been discussed in the lifting schemes [14] and [15]. We also prove the similarity still existence if the computational equations are original definition in DWT algorithm. In the computation paths of DWT, an input signal is sent to two filters separately and downsampled by two. We rewrite IDWT equation analogous to DWT formulae. Hence, both DWT and IDWT can be realized in the same architecture and we just change the filters' coefficients. The manifestation of formulae is suitable for orthogonal and biorthogonal filters. The chosen wavelet filters in our architecture should be a Type I linear phase FIR. The EIC has the following five advantages:

1) the number of multiplication in DWT and IDWT computations are reduced;
2) the instructions can be drawn up regularly and performed by the look-up-table;
3) we adopt parallel organization to speed the execution;
4) both DWT and IDWT computations can be built in the same VLSI architecture;
5) our work needs a fewer number of multipliers and adders, regardless the wavelet filters' length.

The rest of the paper is organized as follows. Section II discusses the basic formulae of the DWT and IDWT, followed by Section III, which focuses on specific instructions that EIC uses. In Section IV, the derivation of the similarity between the 1-D DWT and IDWT computation is shown mathematically. Furthermore, the experimental results and comparisons are listed in Section V. The conclusion of the paper is found in Section VI.

II. Formulae and Notations of the DWT

Both DWT and IDWT can be implemented with filter banks [1], [6]. The 1-D DWT computations are shown in (1), and 1-D IDWT is described in (2)

$$a_1[n] = \sum_k h[k]a_0[2n-k];$$
$$d_1[n] = \sum_k g[k]a_0[2n-k]. \quad (1)$$
$$\hat{a}_0[n] = \sum_k \tilde{h}[n-2k]a_1[k] + \sum_k \tilde{g}[n-2k]d_1[k]. \quad (2)$$

In (1), $a_0[n]$ is the original input signal with a finite length $0 \le n < N$ and its low- and high-pass signals are $a_1[n]$ and $d_1[n]$, respectively. $h[n]$ and $g[n]$ are the low- and high-pass filters of 1-D DWT, respectively. In (2), $\hat{a}_0[n]$ is the reconstructive signal from $a_1[n]$ and $d_1[n]$. In addition, $\tilde{h}[n]$ and $\tilde{g}[n]$ are the low- and high-pass filters of the 1-D IDWT, correspondingly. If $\hat{a}_0[n]$ is required to be equal to $a_0[n]$, the filters of 1-D DWT and 1-D IDWT should have the PR condition [1], [6], [8] and [19]. When a linear-phase signal convolutes with a linear phase filter, it will generate a linear phase output [18]. Therefore, we extend the input sequences symmetrically in the boundary, to make sure that the output signal is symmetric in the boundary under linear phase filters. In this paper, we adopt the wavelet filters (Spline97) with nine-order and seven-order in the low- and high-pass filters, individually [6]. More precisely, the chosen wavelet filters in our architecture should be whole-point symmetric, or a Type I linear-phase finite impulse response (FIR). Spline97 has been modified according to the symmetry property in convolution [18] and [19]. The four filters have the linear phase property

$$\{h[n] \mid -4 \le n \le 4, h[-n] = h[n]\};$$
$$\{g[n] \mid (-1)^n \tilde{h}[1-n], -2 \le n \le 4, g[2-n] = g[n]\}. \quad (3)$$
$$\{\tilde{h}[n] \mid -3 \le n \le 3, \tilde{h}[-n] = \tilde{h}[n]\};$$
$$\{\tilde{g}[n] \mid (-1)^n h[1+n], -5 \le n \le 3, \tilde{g}[-2-n] = \tilde{g}[n]\}. \quad (4)$$

The filter $\tilde{g}[n]$ is not the same as the original definition, since the reconstruction equation in [6] does not adopt the convolution form (2). If the reconstruction equation is rewritten by the convolution form, then (4) would be held. Moreover, based on [18] and [19], the four filters are whole-point symmetric. If the input sequence $a_0[n]$ is extended with whole-point symmetry in the beginning and the end

$$a_0[-n] = a_0[n]; a_0[N-1+n] = a_0[N-1-n] \quad (5)$$

then $a_1[n]$ will become whole-point symmetry in the beginning and half-point symmetry in the end. Conversely, $d_1[n]$ is half-point symmetric in the beginning and whole-point symmetric in the end

$$a_1[-n] = a_1[n]; a_1\left[\frac{N}{2} - 1 + n\right] = a_1\left[\frac{N}{2} - n\right]$$
$$d_1[-n] = d_1[n+1]; d_1\left[\frac{N}{2} + n\right] = d_1\left[\frac{N}{2} - n\right]. \quad (6)$$

Since the arithmetic operations of 1-D DWT are addition and multiplication, this indicates that we can use the simple instructions to complete the operations. The multiplication is executed by a MUL instruction; the addition is done by an ADD instruction. Therefore, we propose that, instruction codes be saved to ROM or RAM in advance for the execution of computation later. We also propose to have GPRs and save the filters' coefficients in ROM or RAM. The MUL instruction is manipulated as follows.

A. MUL Coefficient

Although the multiplication needs specific input and coefficient, we hide the former in MUL. The reason is that the coefficient can be specified by input in order. Once we obtain the sequential input, we only need to arrange the order of coefficients and fetch them one by one. In addition, the product should be saved in a fixed product register. The ADD instruction is manipulated as follows.

B. ADD Specified-Register

All the ADD instruction needs to do is to specify a register to be an accumulator and the value of the accumulator would add up the product and then save the result back to the accumulator.

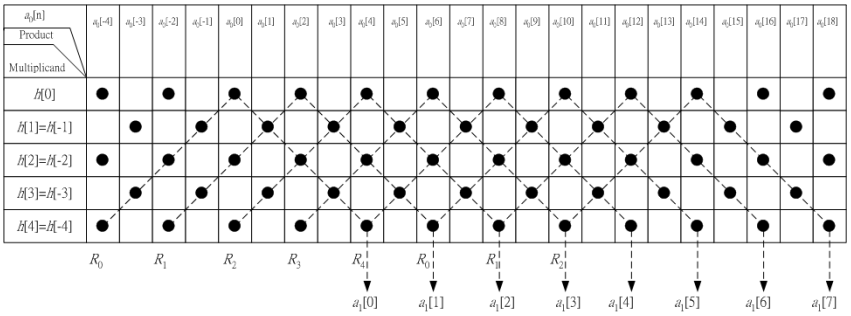

Fig. 1. Computation paths of $a_1[n]$.

TABLE I
MUL AND ADD FUNCTIONS

MUL			ADD		
Instruction	Function		Instruction	Function	
MUL $h[k]$	Input signal multiplies by $h[k]$ and saves the product to PREG		ADD R	PREG adds R and saves the result to R	
Input Item	Instruction	The value of PREG	Input Item	Instruction	The value of R_2
0	MUL $h[4]$	$h[4]a_0[0]$	0	ADD R_2	$R_2 = h[4]a_0[0]$
1	MUL $h[3]$	$h[3]a_0[1]$	1	ADD R_2	$R_2 = (h[4]a_0[0]) + h[3]a_0[1]$
2	MUL $h[2]$	$h[2]a_0[2]$	2	ADD R_2	$R_2 = ((h[4]a_0[0]) + h[3]a_0[1]) + h[2]a_0[2]$
3	MUL $h[1]$	$h[1]a_0[3]$	3	ADD R_2	$R_2 = ((h[4]a_0[0]) + h[3]a_0[1]) + h[2]a_0[2] + h[1]a_0[3]$
⋮	⋮	⋮	⋮	⋮	⋮

III. 1-D DWT Computation

Before we discuss the 1-D DWT computation, we need to set up the nine GPRs $\{R_0, R_1, R_2, \ldots, R_8\}$. The registers R_0, R_1, R_2, R_3 and R_4 are related to $a_1[n]$. And the registers R_5, R_6, R_7 and R_8 are related to $d_1[n]$. Furthermore, the length of the filter decides numbers of registers and a whole-point symmetric filter has odd filter's length. If the length is L, then the numbers of registers are

$$ceil\left(\frac{L}{2}\right). \tag{7}$$

We would explain the equation by following example.

A. Computation of $a_1[n]$

Fig. 1 indicates the computation paths of $a_1[n]$, with the assumption $N = 16$. In the figure, each black dot represents the product obtained from the corresponding $a_0[n]$ indicated above and it multiplies the associated multiplicand $h[k]$ indicated in the column on the far left. This calculation can be done within the single MUL instruction. To obtain each $a_1[n]$, we need to sum up the product represented by the black dots in dash-lines. For example, we can calculate $a_1[2]$ via (1) and obtain

$$\begin{aligned} a_1[2] =& h[4]a_0[0] + h[3]a_0[1] + h[2]a_0[2] \\ &+ h[1]a_0[3] + h[0]a_0[4] + h[-1]a_0[5] \\ &+ h[-2]a_0[6] + h[-3]a_0[7] + h[-4]a_0[8]. \end{aligned}$$

The $a_1[2]$ can be attained by summing up each product generated from the particular $h[k]$ and $a_0[n]$. The black dots of $h[4]a_0[0]$, $h[3]a_0[1]$, $h[2]a_0[2]$, $h[1]a_0[3]$ and $h[0]a_0[4]$ are within one straight dash-line and the dots of $h[-1]a_0[5]$, $h[-2]a_0[6]$, $h[-3]a_0[7]$ and $h[-4]a_0[8]$ are within another dash-line. The summation paths of $a_1[2]$ are merged in two dash-lines. Furthermore, the instructions shown in Table I will complete the calculations and each product will be saved to the product register (PREG).

In the summation path, we need to specify the particular register illustrated at the bottom of Fig. 1, in order to add up the PREG. For example, Fig. 1 shows the summation path of $a_1[2]$, starting with $h[4]a_0[0]$. By tracing the path, the value of R_2, is shown in Table I. When the eighth input is filled, the value of R_2 would be $a_1[2]$. We would output the value of R_2 and zero R_2 for the remaining computation.

From Fig. 1, we can analyze the relationship between the register's numbers and the filter's length. At first, R_0 is the accumulator of $a_1[0]$. After the input $a_0[4]$ is calculated, R_0 is free for the next calculation. Thus, we can utilize the usage of the register to estimate the numbers of registers. Given a filter $\{r[n] \mid L_1 \le n \le L_2, L_2 - L_1 + 1 = L\}$, if we replace $h[k]$ by $r[k]$ in (1), we can assign register R_0 for $a_1[m]$ and $a_1[m]$ can be computed with $\{a_0[2m-L_2], a_0[2m-L_2+1], \ldots, a_0[2m-L_1]\}$. Likewise, R_1 is for $a_1[m+1]$, R_2 is for $a_1[m+2]$ and so on. Register R_p is assigned to $a_1[m+p]$ and $a_1[m+p]$ would be obtained with $\{a_0[2m+2p-L_2], a_0[2m+2p-L_2+1], \ldots, a_0[2m+2p-L_1]\}$. If the first input $a_0[2m+2p-L_2]$ of $a_1[m+p]$ is behind the last input $a_0[2m-L_1]$ of $a_1[m]$, R_0 can be free and replace the function of R_p. It implies $2m - L_1 < 2m + 2p - L_2$. Since $L_2 - L_1 + 1 = L$, we can select the least integer value of P using (7). If R_0 can replace R_p, then R_1 can replace R_{p+1} and so on. Hence, the value of P can be the numbers of registers.

B. Embedded Instruction Code

The EIC is proposed to form the instruction codes and the instructions are the control commands of the PALU organization. The EIC systematically forms the codes in two steps: 1) to observe the values of each register depended on the current input and 2) and check the reuse in PREG.

1) The Values of Each Register: At first, the relationship between $R_0 \sim R_4$ and $a_1[n]$ should be established according to each input. From (1) and Fig. 1, we can list the values of each register, which vary with the input in Table II.

TABLE II
VALUES OF THE REGISTERS

Register / Value / n	$R_0[n]$	$R_1[n]$	$R_2[n]$	$R_3[n]$	$R_4[n]$	Output
0	$a_0[0]\,h[0]$	$a_0[0]\,h[2]$	$a_0[0]\,h[4]$	0	0	No output
1	$R_0[0]+2a_0[1]\,h[1]$	$R_1[0]+a_0[1]\,h[1]$ $+a_0[1]\,h[3]$	$R_2[0]+a_0[1]\,h[3]$	0	0	No output
2	$R_0[1]+2a_0[2]\,h[2]$	$R_1[1]+a_0[2]\,h[0]$ $+a_0[2]\,h[4]$	$R_2[1]+a_0[2]\,h[2]$	$a_0[2]\,h[4]$	0	No output
3	$R_0[2]+2a_0[3]\,h[3]$	$R_1[2]+a_0[3]\,h[1]$	$R_2[2]+a_0[3]\,h[1]$	$R_3[2]+a_0[3]\,h[3]$	0	No output
4	$R_0[3]+2a_0[4]\,h[4]$	$R_1[3]+a_0[4]\,h[2]$	$R_2[3]+a_0[4]\,h[0]$	$R_3[3]+a_0[4]\,h[2]$	$A_0[4]\,h[4]$	$a_1[0] <= R_0[4]$
5	$R_0[5]=0$	$R_1[4]+a_0[5]\,h[3]$	$R_2[4]+a_0[5]\,h[1]$	$R_3[4]+a_0[5]\,h[1]$	$R_4[4]+a_0[5]\,h[3]$	No output
6	$a_0[6]\,h[4]$	$R_1[5]+a_0[6]\,h[4]$	$R_2[5]+a_0[6]\,h[2]$	$R_3[5]+a_0[6]\,h[0]$	$R_4[5]+a_0[6]\,h[2]$	$a_1[1] <= R_1[6]$
7	$R_0[6]+a_0[7]\,h[3]$	$R_1[7]=0$	$R_2[6]+a_0[7]\,h[3]$	$R_3[6]+a_0[7]\,h[1]$	$R_4[6]+a_0[7]\,h[1]$	No output
⋮	⋮	⋮	⋮	⋮	⋮	⋮
12	$R_0[11]+a_0[12]\,h[2]$	$R_1[11]+a_0[12]\,h[0]$	$R_2[11]+a_0[12]\,h[2]$ $+a_0[12]\,h[4]$	N/A	$R_4[11]+a_0[12]\,h[4]$	$a_1[4] <= R_4[12]$
13	$R_0[12]+a_0[13]\,h[3]$	$R_1[12]+a_0[13]\,h[1]$	$R_2[12]+a_0[13]\,h[1]$ $+a_0[13]\,h[3]$	N/A	N/A	No output
14	$R_0[13]+a_0[14]\,h[4]$	$R_1[13]+a_0[14]\,h[2]$ $+a_0[14]\,h[4]$	$R_2[13]+a_0[14]\,h[0]$ $+a_0[14]\,h[2]$	N/A	N/A	$a_1[5] <= R_0[14]$
15	N/A	$R_1[14]+a_0[15]\,h[3]$	$R_2[14]+a_0[15]\,h[1]$	N/A	N/A	$a_1[6] <= R_1[16]$ $a_1[7] <= R_3[18]$

TABLE III
NUMBER OF MULTIPLICATIONS

Method	The number of multiplication	Interpretation
Mathematics	$9N/2$	According to (1) and the chosen filter, it needs to execute 9 multiplications for each $a_1[n]$. Since $0 \le n < N/2$, the number of multiplication is $9N/2$.
Reuse the PREG	$5N/2$	Fig. 1 has shown the products that can be used in computation of $a_1[n]$, and they are marked by the black dot. Thus, the number of the black dots in Fig. 1 is the maximum number of multiplication while we compute $a_1[n]$. Since we have dealt with the boundary, the black dots obtained from $a_0[-4]$ to $a_0[-1]$ and from $a_0[16]$ to $a_0[18]$ can be omitted. Consequently, the number of multiplication is $3(N/2)+2(N/2)=5N/2$.

The symbol "<=" means that the right operand is transmitted to the left operand. Additionally, "N/A" means that the register will no longer be used. Moreover, through (5) we can assign the value of the boundary inputs: $n < 0$ and $n > 15$. Since the value of PREG needs to be multiplied by two within the boundary $0 \le n \le 4$, we add a least-significant bit shift (LS) preceding the adder in the architecture, to be activated by a multiplexer. Table II also shows the relationship between inputs and outputs. $a_1[n]$ can be obtained with $a_0[2n+4]$ and $a_0[2n+5]$ if $4 \le 2n+4, 2n+5 \le N-3$ and $\{a_1[N/2-1], a_1[N/2-2], a_1[N/2-3]\}$ can be acquired if $\{a_0[n] \mid N-2 \le n \le N-1\}$. Furthermore, there will not be any output if $\{a_0[n] \mid 0 \le n \le 3\}$.

2) The Reuse in PREG: Table II reveals that some products could be shared by registers. Take $n = 4$, for example; $h[4]a_0[4]$ will be accumulated to R_0 and R_4 and $h[2]a_0[4]$ will be accumulated to R_1 and R_3. Thus, we can use one MUL and accumulate the PREG to two different registers with two ADDs. In Fig. 1, the products, represented by black dots, are shared by two registers as two dash-lines cross one another. The exceptions are the products in the second row, whose multiplicand is $h[0]$. Since the product sharing is based on the symmetry property, we would build one multiplier and two adders (1M2A) in PALU for $a_1[n]$ computation. Hence, since there are nine multiplications for each $a_1[n]$, our method can cut down on the number of multiplications. Table III lists the multiplication reduction.

3) Instruction Codes for the Computation of $a_1[n]$: From Table II, we can describe instruction codes in Table IV that can perform the calculations of $a_1[n]$ and the explanation of the table as follows.

TABLE IV
INSTRUCTION CODES OF $a_1[n]$

Boundary in the beginning			Loop			Boundary in the end		
Input item	Order	Instruction	Input item	Order	Instruction	Input item	Order	Instruction
0	0	MUL $h[0]$;	6	30	MUL $h[0]$;	12	60	MUL $h[0]$;
	1	ADD R_0;		31	ADD R_3;		61	ADD R_1;
	2	MUL $h[2]$;		32	MUL $h[2]$;		62	MUL $h[2]$;
	3	ADD R_1;		33	ADD R_2; ADD R_4;		63	ADD R_0; ADD R_2;
	4	MUL $h[4]$;		34	MUL $h[4]$;		64	MUL $h[4]$;
	5	ADD R_2;		35	ADD R_0; ADD R_1;		65	ADD R_2; ADD R_4;
1	6	MUL $h[1]$;	7	36	MUL $h[1]$;	13	66	MUL $h[1]$;
	7	ADD R_0(LS); ADD R_1;		37	ADD R_3; ADD R_4;		67	ADD R_1; ADD R_2;
	8	MUL $h[3]$;		38	MUL $h[3]$;		68	MUL $h[3]$;
	9	ADD R_1; ADD R_2;		39	ADD R_0; ADD R_2;		69	ADD R_2; ADD R_0;
2	10	MUL $h[0]$;	8	40	MUL $h[0]$;	14	70	MUL $h[0]$;
	11	ADD R_1;		41	ADD R_4;		71	ADD R_2;
	12	MUL $h[2]$		42	MUL $h[2]$;		72	MUL $h[2]$;
	13	ADD R_0(LS); ADD R_2;		43	ADD R_3; ADD R_0;		73	ADD R_1; ADD R_2;
	14	MUL $h[4]$;		44	MUL $h[4]$;		74	MUL $h[4]$;
	15	ADD R_1; ADD R_3;		45	ADD R_1; ADD R_2;		75	ADD R_1; ADD R_0;
3	16	MUL $h[1]$;	9	46	MUL $h[1]$;	15	76	MUL $h[1]$;
	17	ADD R_1; ADD R_2;		47	ADD R_4; ADD R_0;		77	ADD R_2;
	18	MUL $h[3]$;		48	MUL $h[3]$;		78	MUL $h[3]$;
	19	ADD R_0(LS); ADD R_3;		49	ADD R_1; ADD R_3;		79	ADD R_1;
4	20	MUL $h[0]$;	10	50	MUL $h[0]$;			
	21	ADD R_2;		55	ADD R_0;			
	22	MUL $h[2]$;		52	MUL $h[2]$;			
	23	ADD R_1; ADD R_3;		53	ADD R_4; ADD R_1;			
	24	MUL $h[4]$;		54	MUL $h[4]$;			
	25	ADD R_0(LS); ADD R_4;		55	ADD R_2; ADD R_3;			
5	26	MUL $h[1]$;	11	56	MUL $h[1]$;			
	27	ADD R_2; ADD R_3;		57	ADD R_0; ADD R_1;			
	28	MUL $h[3]$;		58	MUL $h[3]$;			
	29	ADD R_1; ADD R_4;		59	ADD R_2; ADD R_4;			

TABLE V
REGULATION IN "LOOP" OF $a_1[n]$

Input item	Instruction	Relation	Input item	Instruction	Relation
$n \in even$ and $6 \le n \le N-5$	MUL $h[0]$;	$u(m,n)$: the remainder of $(n/2+m)/5$, where $m \in [0,1,2,3,4]$.	$n \in odd$ and $6 \le n \le N-5$	MUL $h[1]$;	$u(m,n)$: the remainder of $((n-1)/2+m)/5$, where $m \in [0,1,2,4]$.
	ADD $R_{u(0,n)}$;			ADD $R_{u(0,n)}$; ADD $R_{u(1,n)}$;	
	MUL $h[2]$;			MUL $h[3]$;	
	ADD $R_{u(4,n)}$; ADD $R_{u(1,n)}$;			ADD $R_{u(2,n)}$; ADD $R_{u(4,n)}$;	
	MUL $h[4]$;				
	ADD $R_{u(3,n)}$; ADD $R_{u(2,n)}$;				

- The "Order" indicates the execution order and the "(LS)" marks that the LS be activated. In some execution order, there exist two instructions that should be executed concurrently. For example, in order 9, we have "ADD R_1" and "ADD R_2". We limit ADD and MUL instructions in the same execution time for PALU organization design.
- There are three classifications in the instruction codes: "Boundary in the beginning", "Loop" and "Boundary in the end". The codes in "Loop" are in a certain sequence, shown in Table V. The instruction codes in "Loop" for the nth input are the same as those belonging to $(n+10)$th, where $6 \le n \le N-5$. Hence, we just record the instructions within $6 \le n \le 15$ and execute them periodically.
- We use a simple version of execution order, shown in Fig. 2, to interpret the period in "Loop". Each coefficient in Fig. 2 stands for the instructions, e.g., $h[0]$ in the circle represents the instructions when $n = 6$: MUL $h[0]$ and ADD R_3. In addition, mark C is labeled that the register should be cleaned, e.g., $R_1[7] = 0$ in Table II. Furthermore, the coefficients arranged above $n = 16$ is the same as that above $n = 16$. The coefficients allocated above $16 \le n \le 25$ would be the same as that above $6 \le n \le 15$. The reason is that each register should be accumulated nine times and finally zeroed. After that, the register can be reused and the instructions would be executed periodically. The number of the addition is decided by the filter's length. Thus, the period in "Loop" is $L+1$ if the filter's length is L.
- The codes for "Boundary in the end" are dependent on the value of N and are listed in Table VI. Here, $u(m,N)$ is the remainder of $(N/2+m)/5$ and $m \in [1,2,3,4]$.

4) PALU Organization of 1M2A: From Tables IV–VI, our architecture includes the PALU organization, that is formed with one multiplier and two adders (1M2A), shown in Fig. 3. Furthermore, the organization comprises one 16-bit multiplier and two 32-bit adders. To parallelize the organization, we use the latch PREG, which contains the product and the product is obtained from the MUL executed in previous order. Thus, the ADD can employ the previous value and the MUL can generate the new product concurrently.

C. Computation of $d_1[n]$

Likewise, we could apply the EIC and PALU organization to carry out $d_1[n]$ computation. Fig. 4 shows the computation paths of $d_1[n]$. The boundary could be revised by using (3) and the reapplication in PREG should be considered. We show the part of the instruction codes for $d_1[n]$ in Tables VII–IX. Note that $u(m,N)$ in Table VIII is 5 plus the remainder of $(N/2+m)/4$,

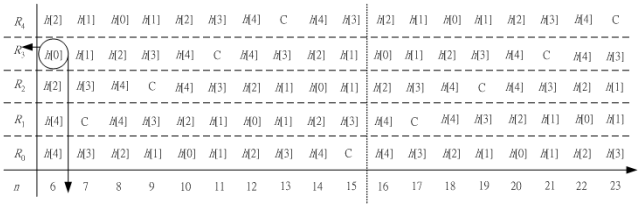

Fig. 2. The period in "Loop."

TABLE VI
INSTRUCTION CODES FOR THE "BOUNDARY IN THE END" OF $a_1[n]$

Input item	Instruction	Input item	Instruction
N-4	MUL $h[0]$;	N-2	MUL $h[0]$;
	ADD $R_{u(3,N)}$;		ADD $R_{u(4,N)}$;
	MUL $h[2]$;		MUL $h[2]$;
	ADD $R_{u(2,N)}$; ADD $R_{u(4,N)}$;		ADD $R_{u(3,N)}$; ADD $R_{u(4,N)}$;
	MUL $h[4]$;		MUL $h[4]$;
	ADD $R_{u(4,N)}$; ADD $R_{u(1,N)}$;		ADD $R_{u(3,N)}$; ADD $R_{u(2,N)}$;
N-3	MUL $h[1]$;	N-1	MUL $h[1]$;
	ADD $R_{u(3,N)}$; ADD $R_{u(4,N)}$;		ADD $R_{u(4,N)}$;
	MUL $h[3]$;		MUL $h[3]$;
	ADD $R_{u(4,N)}$; ADD $R_{u(2,N)}$;		ADD $R_{u(3,N)}$;

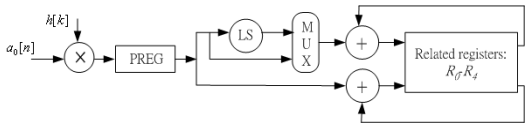

Fig. 3. PALU organization of 1M2A.

where $m \in [5,6,7,8]$. Finally, based on the previous explanation of PALU, we allocate another 1M2A for $d_1[n]$ computation. Thus, the eventual organization would be called 2M4A PALU.

IV. 1-D IDWT COMPUTATION

In 1-D IDWT, we transform the original 1-D IDWT equation (2) into new equations, and the equations are similar to 1-D DWT equations (1), according to the theories of multirate systems [1]. At first, we individually divide $\hat{a}_0[n]$, $\tilde{H}(z)$, and $\tilde{G}(z)$ into two parts: the even part and odd part

$$\begin{aligned} \hat{a}_{0e}[n] &= \hat{a}_0[2n]; \hat{a}_{0o}[n] = \hat{a}_0[2n+1] \\ \hat{A}_0(z) &= \hat{A}_{0e}\left(z^2\right) + \hat{A}_{0o}\left(z^2\right) z^{-1} \qquad (8) \\ \tilde{H}(z) &= \tilde{H}_e\left(z^2\right) + \tilde{H}_o(z^2) z^{-1} \\ \tilde{G}(z) &= \tilde{G}_e\left(z^2\right) + \tilde{G}_o\left(z^2\right) z^{-1} \qquad (9) \end{aligned}$$

where

$$\tilde{H}_e(z) = \sum_k \tilde{h}[2k] z^{-k}, \quad \tilde{H}_o(z) = \sum_k \tilde{h}[2k+1] z^{-k}$$
$$\tilde{G}_e(z) = \sum_k \tilde{g}[2k] z^{-k}, \quad \tilde{G}_o(z) = \sum_k \tilde{g}[2k+1] z^{-k}.$$

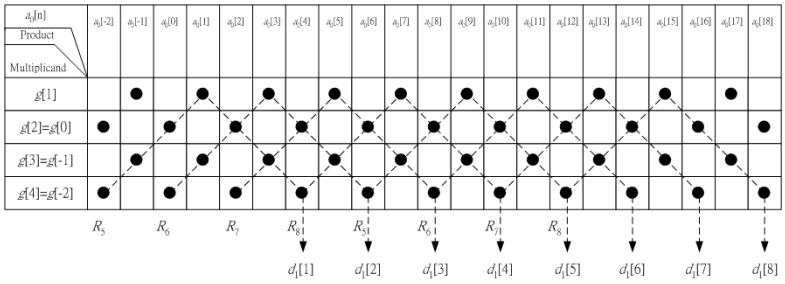

Fig. 4. Computation paths of $d_1[n]$.

TABLE VII
INSTRUCTION CODES FOR THE "BOUNDARY IN THE BEGINNING" OF $d_1[n]$

Input item	Instruction	Input item	Instruction
0	MUL $g[2]$;	2	MUL $g[2]$;
	ADD R_5;		ADD R_5;
			ADD R_6;
	MUL $g[4]$;		MUL $g[4]$;
	ADD R_6;		ADD R_7;
			ADD R_5;
1	MUL $g[1]$;	3	MUL $g[1]$;
	ADD R_5;		ADD R_6;
	MUL $g[3]$;		MUL $g[3]$;
	ADD R_6;		ADD R_5;
	ADD R_5;		ADD R_7;

TABLE VIII
INSTRUCTION CODES FOR THE "BOUNDARY IN THE END" OF $d_1[n]$

Input item	Instruction	Input item	Instruction
N-4	MUL $g[2]$;	N-2	MUL $g[2]$;
	ADD $R_{u(5,N)}$;		ADD $R_{u(7,N)}$ (LS);
	ADD $R_{u(6,N)}$;		ADD $R_{u(6,N)}$;
	MUL $g[4]$;		MUL $g[4]$;
	ADD $R_{u(7,N)}$ (LS);		ADD $R_{u(5,N)}$;
	ADD $R_{u(8,N)}$;		ADD $R_{u(6,N)}$;
N-3	MUL $g[1]$;	N-1	MUL $g[1]$;
	ADD $R_{u(6,N)}$;		ADD $R_{u(7,N)}$;
	MUL $g[3]$;		MUL $g[3]$;
	ADD $R_{u(7,N)}$ (LS);		ADD $R_{u(6,N)}$;
	ADD $R_{u(5,N)}$;		

Furthermore, from the 1-D IDWT equation (2), we get

$$\begin{aligned}\hat{A}_0(z) =&\tilde{H}(z)A_1(z^2) + \tilde{G}(z)D_1(z^2)\\ =&\{A_1(z^2)\tilde{H}_e(z^2) + D_1(z^2)\tilde{G}_e(z^2)\}\\ &+ z^{-1}\left\{A_1(z^2)\tilde{H}_o\left(z^2\right) + D_1\left(z^2\right)\tilde{G}_o\left(z^2\right)\right\}.\end{aligned} \tag{10}$$

Then, we introduce a new signal $w_1[n]$, which is alternately composed of $a_1[n]$ and $d_1[n]$ as follows:

$$\begin{aligned}w_1[2n] =&a_1[n]; \quad w_1[2n-1] = d_1[n]\\ W_1(z) =&A_1\left(z^2\right) + D_1\left(z^2\right)z.\end{aligned} \tag{11}$$

From (6), we see that $w_1[n]$ belongs to the whole-point symmetry in the beginning and the end. In the following, if $W_1(z)$ is multiplied by two specific filters defined in (12), we would get (13) and (14)

$$\begin{aligned}B_L(z) =&\tilde{H}_e\left(z^2\right) + \tilde{G}_e\left(z^2\right)z^{-1}\\ B_H(z) =&\tilde{H}_o\left(z^2\right) + \tilde{G}_o\left(z^2\right)z^{-1}\\ W_1(z)B_L(z) =&W_1(z)\left(\tilde{H}_e\left(z^2\right) + \tilde{G}_e\left(z^2\right)z^{-1}\right)\end{aligned} \tag{12}$$

$$\begin{aligned}=&\hat{A}_{0e}\left(z^2\right) + \left\{A_1\left(z^2\right)\tilde{G}_e\left(z^2\right)z^{-1}\right.\\ &\left. + D_1\left(z^2\right)\tilde{H}_e\left(z^2\right)z\right\}\end{aligned} \tag{13}$$

$$\begin{aligned}W_1(z)B_H(z) =&W_1(z)\left(\tilde{H}_o\left(z^2\right) + \tilde{G}_o\left(z^2\right)z^{-1}\right)\\ =&\hat{A}_{0o}\left(z^2\right) + \left\{A_1\left(z^2\right)\tilde{G}_o\left(z^2\right)z^{-1}\right.\\ &\left. + D_1\left(z^2\right)\tilde{H}_o\left(z^2\right)z\right\}.\end{aligned} \tag{14}$$

Eventually, we respectively downsample the signals (13) and (14) by two and get (15), as follows:

$$\hat{a}_{0e}[n] = \sum_k b_L[k]w_1[2n-k]; \hat{a}_{0o}[n] = \sum_k b_H[k]w_1[2n-k]. \tag{15}$$

It is clear that the computation path of the 1-D IDWT is similar to the 1-D DWT. Additionally, because of the symmetry property in (4), (16) is supported

$$\tilde{H}(z) = \tilde{H}\left(z^{-1}\right); \tilde{G}(z) = z^2\tilde{G}\left(z^{-1}\right). \tag{16}$$

Besides, from (16), we could attain (17), which also have the symmetry property

$$\begin{aligned}\tilde{H}_e\left(z^2\right) =&\tilde{H}_e\left(z^{-2}\right), \quad \tilde{H}_o\left(z^2\right) = z^2\tilde{H}_o\left(z^{-2}\right)\\ \tilde{G}_e\left(z^2\right) =&z^2\tilde{G}_e\left(z^{-2}\right), \quad \tilde{G}_o\left(z^2\right) = z^4\tilde{G}_o\left(z^{-2}\right).\end{aligned} \tag{17}$$

Using (17), we can examine the symmetry property of $B_L(z)$ and $B_H(z)$ in (18)

$$B_L\left(z^{-1}\right) = B_L(z); B_H\left(z^{-1}\right) = z^{-2}B_H(z). \tag{18}$$

Thus, due to linear phase [17] and Spline97, $B_L(z)$ and $B_H(z)$ belong to the linear-phase and whole-point symmetric filter

$$\begin{aligned}&\{b_L[n] \mid -3 \le n \le 3, b_L[-n] = b_L[n]\}\\ &\{b_H[n] \mid -5 \le n \le 3, b_H[-2-n] = b_H[n]\}.\end{aligned} \tag{19}$$

Based on the discussions, the computation paths of $\hat{a}_{0e}[n]$ and $\hat{a}_{0o}[n]$ are demonstrated in Figs. 5 and 6, respectively. Thus, EIC can also be used in 1-D IDWT and we just need to change the filters' coefficients and instruction codes. The VLSI architecture and PALU organization for 1-D DWT still applies in 1-D IDWT.

V. EXPERIMENTAL RESULTS AND COMPARSIONS

We accomplish the 2-D DWT by applying 1-D DWT in column and row (separable approach) and the PALU organization in 1-D DWT architecture is adopted 2M4A. In addition,

TABLE IX
REGULATION IN"LOOP" OF $d_1[n]$

Input item	Instruction	Relation	Input item	Instruction	Relation
$n \in even$ and $4 \le n \le N-5$	MUL $g[2]$; ADD $R_{u(7,n)}$; ADD $R_{u(8,n)}$; MUL $g[4]$; ADD $R_{u(6,n)}$; ADD $R_{u(5,n)}$;	$u(m,n)$:5 plus the remainder of $(n/2+m)/4$, where $m \in [5,6,7,8]$.	$n \in odd$ and $4 \le n \le N-5$	MUL $g[1]$; ADD $R_{u(8,n)}$; MUL $g[3]$; ADD $R_{u(5,n)}$; ADD $R_{u(7,n)}$;	$u(m,n)$:5 plus the remainder of $((n-1)/2+m)/4$, where $m \in [5,7,8]$.

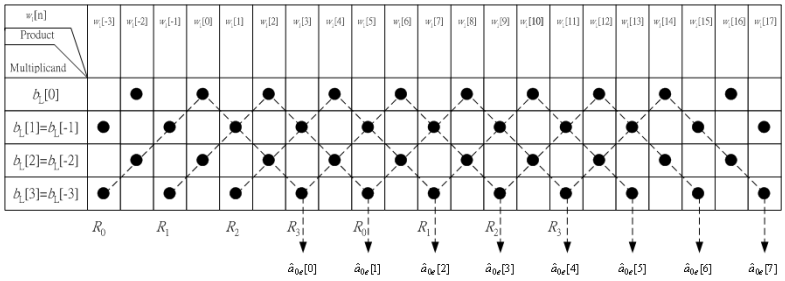

Fig. 5. Computation paths of $\hat{a}_{0e}[n]$.

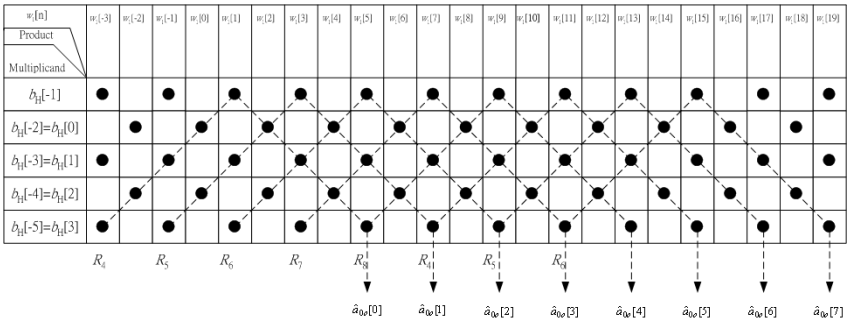

Fig. 6. Computation paths of $\hat{a}_{0o}[n]$.

TABLE X
PROPERTIES AND SPECIFICATION OF THE CHIP

TSMC 0.35μm 1P4M CMOS technology.
Pin counts are 84 pins.
Operation frequency is 62.5MHz for worse case and 100MHz for normal case.
Processing speed is 4.88Mpixel/sec for worse case and 7.78Mpixel/sec for normal case.
Gate counts are 30192 gates.
The gate counts of data path are 4669 gates.
The gate counts of 1-D control are 12644 gates.
The gate counts of 2-D control are 10309 gates.
The gate counts of I/O are 3290 gates.
Core size is $1697\mu m \times 1722\mu m$.
Die size is $3123\mu m \times 3102\mu m$.
The functions include 2-D DWT and IDWT. The decomposition level is up to 6.

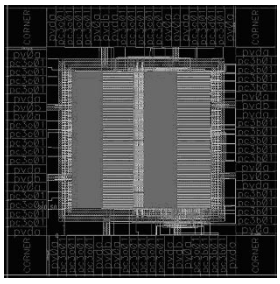

Fig. 7. Layout view of the chip.

we employ VHDL language to describe the circuits and the layout synthesis is done with the CADENCE layout tools and the TSMC 0.35-μm 1P4M CMOS technology. We test the chip with the 2-D signal and each pixel of the testing signal is 8 bits. Furthermore, we could select the width or height of the 2-D image signals among five groups of parameters: 1) 1024, 512, 256, 128, 64, 32; 2) 640, 320, 160, 80, 40; 3) 480, 240, 120, 60, 30; 4) 352, 176, 88, 44; and 5) 720, 360, 180, 90 pixels. Therefore, our chip can support the image size of VGA, MPEG-1, and MPEG-2 [25] and up to 1025×1025. Table X indicates the properties and specification. Finally, we show the layout in Fig. 7.

The recent researches [20]–[22], are concluded to compare in Table XI. In [20], their architecture sustains 512×512 image size and the size is smaller than that of ours. In addition, they do not show the precision bits. On the other hand, in [21], despite their precision bits are the same as ours, our design enjoys superiority in supported image size with specifications like VGA, MPEG-1, and MPEG-2. Moreover, in [22], though their wavelet filters are programmable, their doable image size is 256×256 and the precision is 24 bits. Thus, our chip has advantages in precision and image size.

The merits of our work are summarized in the following. Our chip integrates DWT and IDWT into a single chip without extra circuits and uses fewer pin counts at just 84 pins. The goals

TABLE XI
COMPARISONS

Method	Sheu *et al.* [20]	Lee *et al.* [21]	Chen *et al.* [22]	Ours
Architecture highlight	Separable	Separable	Programmable filter length	Separable
IDWT	No	No	No	Yes
Process (μm)	0.6	0.6	0.35	0.35
Total area ($\mu m \times \mu m$)	7600×8400	5344.8×5287.2	5200×2500	3123×3102
operation frequency (MHz)	25	100(external) 50(internal)	50	100
image size (pixel×pixel)	512×512	512×512	256×256	1024×1024
other image size (pixel)	No	No	No	640×480 (VGA) 352×240 (MPEG-1) 720×480 (MPEG-2)
Precision bits	N/A	32	24	32

*Input signal is 8-bit form and output signal is 12-bit form

of lower hardware cost and architecture sharing are therefore achieved. The decomposition level is up to six and the image sizes are in many ways better supported, with contemporary applications' specifications.

VI. CONCLUSION

In this paper, we propose VLSI architecture and a design regulation named, EIC, to perform the 2-D DWT/IDWT. We establish a set of instructions to accomplish the DWT computation. In addition, we present a PALU organization for the computation unit and it consists of two multipliers and four adders. Since the similarity between DWT and IDWT is proven, we can build both DWT and IDWT in the same architecture. Moreover, we employ symmetric extension to satisfy the PR requirement. Using the TSMC 0.35 μm 1P4M CMOS technology, our experiment is 3123 μm $\times$ 3102 μm in the total area, the pin counts at 84 pins and the processing speed at 7.78 Mpixels/s. Finally, our work supports up to six decomposition levels and versatile image specifications.

ACKNOWLEDGMENT

The authors express great appreciation to Dr. C.-Y. Su and J.-W. Wang for their kind help and valuable opinion, dedicated into the research.

REFERENCES

[1] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*. Englewood Cliffs, NJ: Prentice-Hall, 1993, pp. 491–538.
[2] M. Vishwanath, R. M. Owens, and M. J. Irwin, "VLSI architectures for the discrete wavelet transform," *IEEE Trans. Circuits Syst. II*, vol. 42, pp. 305–316, May 1995.
[3] M. Ferretti and D. Rizzo, "Handling borders in systolic architectures for the 1-D discrete wavelet transform for perfect reconstruction," *IEEE Trans. Signal Processing*, vol. 48, pp. 1365–1378, May 2000.
[4] M. Vishwanath, "The recursive pyramid algorithm for the discrete wavelet transform," *IEEE Trans. Signal Processing*, vol. 42, pp. 673–677, Mar. 1994.
[5] JPEG 2000. [Online]. Available: http://www.jpeg.org/
[6] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transform," *IEEE Trans. Image Processing*, vol. 1, pp. 205–220, Apr. 1992.
[7] G. Knowles, "VLSI architecture for the discrete wavelet transform," *Electron. Lett.*, vol. 26, no. 15, pp. 1184–1185, July 1990.
[8] S. G. Mallat, "A theory for multiresolution signal decompositions: The wavelet representation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 11, pp. 674–693, July 1989.
[9] T. C. Denk and K. K. Parh, "Systolic VLSI architectures for 1-D discrete wavelet transforms," in *Proc. 1998 32nd Asilomar Conf. Signals, Systems and Computers*, vol. 2, 1998, pp. 1220–1224.
[10] W. Shunjun, W. Chao, and S. Yong, "A vlsi implementation structure for wavelet decomposition filter," in *Proc. 2000 IEEE Int. Conf. Multimedia and Expo*, vol. 3, 2000, pp. 1399–1402.
[11] T. Acharya and P. Y. Chen, "VLSI implementation of a DWT architecture," in *Proc. IEEE Int. Symp. Circuits and Systems*, May 1998, pp. 272–275.
[12] F. Marino, D. Guevorkian, and J. T. Astola, "Highly efficient high-speed/low-power architectures for the 1-D discrete wavelet transform," *IEEE Trans. Circuits Syst. II*, vol. 47, Dec. 2000.
[13] C. Yu, C.-A. Hsieh, and S.-J. Chen, "Design and implementation of a highly efficient VLSI architecture for discrete wavelet transform," in *Proc. IEEE 1997 Custom Integrated Circuits Conf.*, 1997, pp. 237–240.
[14] K. Andra, C. Chakrabarti, and T. Acharya, "A VLSI architecture for lifting-based wavelet transform," in *2000 IEEE Workshop on Signal Processing Systems*, 2000, pp. 70–79.
[15] 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 *Proc. 2001 IEEE Int. Symp. Circuits and Systems*, vol. 2, 2001, pp. 529–532.
[16] S. Masud and J. V. McCanny, "Rapid VLSI of biorthogonal wavelet transform cores," in *1999 IEEE Workshop on Signal Processing Systems*, 1999, pp. 291–300.
[17] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 1998, pp. 291–311.
[18] G. Strang and T. Nguyen, *Wavelets and Filter Banks*. Wellesley, MA: Wellesley-Cambridge Press, 1996, pp. 272–275.
[19] B.-F. Wu and C.-Y. Su, "A fast convolution algorithm for biorthogonal wavelet image compression," *J. Chinese Inst. Eng.*, vol. 22, no. 2, pp. 179–192, 1999.
[20] M.-H. Sheu, M.-D. Shieh, and S.-W. Liu, "A VLSI architecture design with lower hardware cost and less memory for separable 2-D discrete wavelet transform," in *Proc. 1998 IEEE Int. Symp. Circuits and Systems*, vol. 5, 1998, pp. 457–460.
[21] C.-Y. Lee and W.-S. Peng, "An Efficient Algorithm and Architecture Design for Two-Dimension Separable Discrete Wavelet Transform," Master's thesis, National Chaio Tung Univ., Hsinchu, Taiwan, 1999.
[22] C.-Y. Chen, Z.-L. Yang, T.-C. Wang, and L.-G. Chen, "A programmable parallel VLSI architecture for 2-D discrete wavelet transform," *J. VLSI Signal Processing Syst. Signal Image and Video Technol.*, vol. 28, no. 3, pp. 151–163, July 2001.
[23] G. Lafruit, L. Nachtergaele, J. Borman, M. Engels, and I. Bolsen, "Optimal memory organization for scalable texture codes in MPEG-4," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, pp. 218–243, Apr. 1999.
[24] M. Ferretti and D. Rizzo, "A parallel architecture for the 2-D discrete wavelet transform with integer lifting scheme," *J. VLSI Signal Processing Syst. Signal, Image and Video Technol.*, vol. 28, no. 3, pp. 165–185, 2001.
[25] MPEG website. [Online]. Available: http://www.mpeg.org/