Parallel form of the Pipelined Lifting-based VLSI Architectures for Two-dimensional Discrete Wavelet Transform

Ibrahim Saeed Koko, *Member, IAENG* and Herman Agustiawan

***Abstract*—In this paper, in order to best meet real-time applications of 2-dimensional discrete wavelet transform (2-D DWT) with demanding requirements in terms of speed and throughput, 2-parallel and 4-parallel pipelined lifting-based VLSI architectures for lossless 5/3 and lossy 9/7 algorithms are proposed. The two proposed parallel architectures achieve speedup factors of 2 and 4 as compared with single pipelined architecture based on the first scan method proposed by Ibrahim et al. The advantage of the proposed parallel architectures is that the total temporary line buffer (TLB) does not increase from that of the single pipelined architecture proposed by Ibrahim et al. when degree of parallelism is increased.**

***Index Terms*—VLSI architecture, parallel, discrete wavelets transform (DWT), JPEG2000, and lifting scheme.**

I. Introduction

2-D DWT has evolved as an essential part of a modern compression system such as JPEG2000. It offers high compression ratio with good image quality and overcomes disadvantage of the DCT based compression system, which suffers from blocks artifacts that decrease the quality of the displayed image. In addition, wavelet based compression system, not only presents superior compression performance over DCT, but provides four dimension of scalabilities — resolution, distortion, spatial, and color, which are very difficult to achieve in DCT-based compression system. These superior features make JPEG2000 ideal for use in power and bandwidth limited applications [1].

In a compression system, the function of DWT is to decorrelate the original image pixels prior to compression step such that they can be amenable to compression.

In this paper, to further enhance the performance in terms of speed and throughput in order to best meet real-time applications of 2-D DWT with demanding requirements, 2-parallel and 4-parallel pipelined architectures for 5/3 and 9/7 algorithms are proposed. The 2-parallel and the 4-parallel architectures achieve speedup factors of 2 and 4, respectively, as compared with the single pipelined architecture present in [2]. The two proposed architectures are based on lifting scheme, which facilitates high speed and efficient implementation of wavelet transforms [3].

The authors are with the Electrical and Electronic Engineering Department, Universiti Teknologi PETRONAS, Perak, Tronoh, Malaysia

This paper is organized as follows. In section II, 5/3 and 9/7 algorithms are stated and the data dependency graphs (DDGs) for both algorithms are given. In section III, The proposed architectures are presented. The performance evaluations are given in section IV. Results comparison and conclusions are given in sections V and VI, respectively.

II. Lifting-based 5/3 and 9/7 Algorithms

The lossless 5/3 and lossy 9/7 wavelet transforms algorithms are defined by the JPEG2000 image compression standard as follow [4]:

5/3 analysis algorithm

$$\begin{aligned} &step1: Y(2j+1) = X(2j+1) - \left\lfloor \frac{X(2j)+X(2j+2)}{2} \right\rfloor \\ &step2: Y(2j) = X(2j) + \left\lfloor \frac{Y(2j-1)+Y(2j+1)+2}{4} \right\rfloor \end{aligned} \tag{1}$$

9/7 analysis algorithm

$$\begin{aligned} &step1: Y''(2n+1) = X(2n+1) + a\left(X(2n) + X(2n+2)\right) \\ &step2: Y''(2n) = X(2n) + b\left(Y''(2n-1) + Y''(2n+1)\right) \\ &step3: Y'(2n+1) = Y''(2n+1) + g\left(Y''(2n) + Y''(2n+2)\right) \\ &step4: Y'(2n) = Y''(2n) + d\left(Y'(2n-1) + Y'(2n+1)\right) \\ &step5: Y(2n+1) = 1/k\, Y'(2n+1) \\ &step6: Y(2n) = kY'(2n) \end{aligned} \tag{2}$$

The data dependency graphs (DDGs) for 5/3 and 9/7 derived from the algorithms are shown in figures 1 and 2, respectively. The DDGs are very useful tools in architecture development and they provide the information necessary for the designer to develop more accurate architectures. The symmetric extension algorithm is incorporated in the DDGs to handle the boundaries problems. The boundary treatment is necessary to keep number of wavelet coefficients the same as that of the original [4], and is applied at the beginning and ending of each row or column in an *NxM* image.

III. Proposed Parallel Architectures

To ease the architecture development, the strategy adopted in [2, 5] was to divide the details of the development into two steps each having less information to handle. In the first step, the DDGs were looked at from outside, as indicated by the dotted boxes in Figs. 1 and 2, in terms of inputs and outputs requirements. It was observed that the DDGs for 5/3 and 9/7 are identical when they are looked at from external, taking into consideration only the inputs and outputs requirements but they differ in the internal details. Based on

DOI: 10.7763/IJCTE.2009.V1.14

this observation, the first level of the architecture, called, the external architecture was developed. In the second step, the internal details of the DDGs were considered for the development of the processor's datapath architectures, since DDGs internally define and specify the structure of the processors.

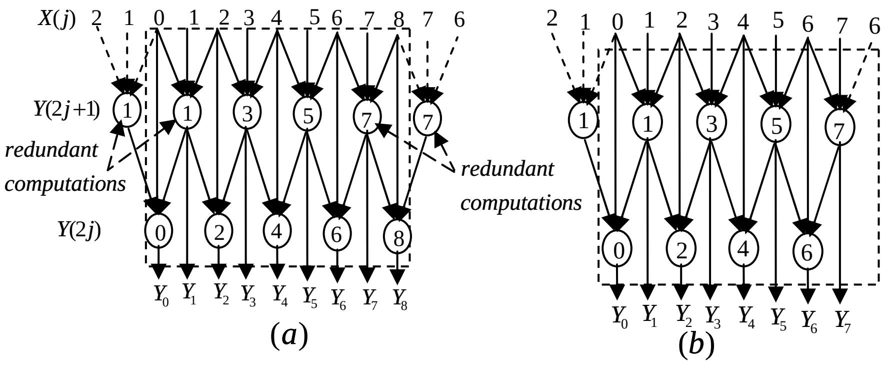

Fig. 1 5/3 algorithm's DDGs for (a) odd and (b) even length signals

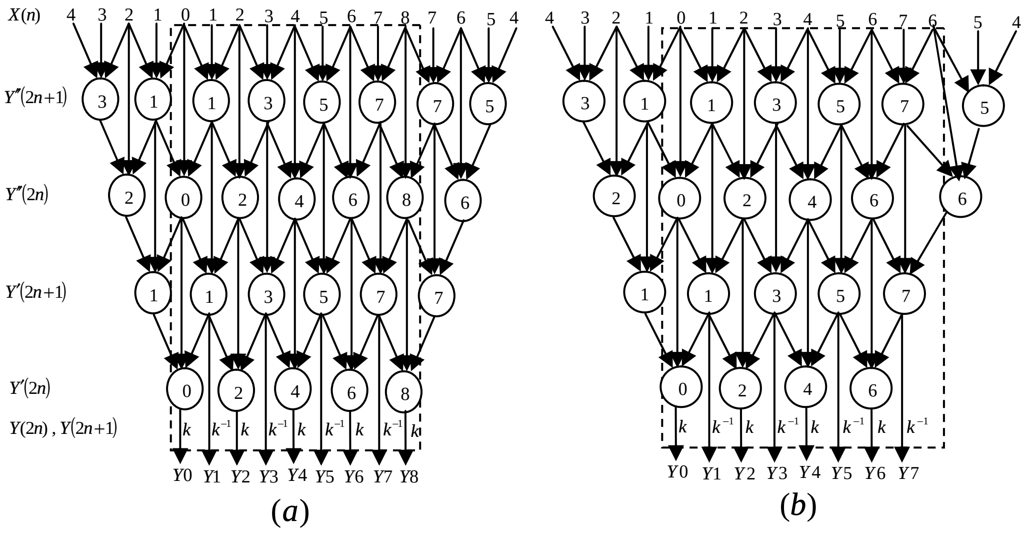

Fig. 2 9/7 algorithm's DDG for odd (a) and even (b) length signals

In this paper, the first level, the external architectures for 2-parallel and 4-parallel are developed for both 5/3 and 9/7 algorithms. Then, processors datapath architectures for 5/3 and 9/7 developed in [2] are modified to fit into RPs and CPs of the parallel architectures.

In general, the scan frequency f_l and hence the period $t_l = 1/f_l$ of the parallel architectures can be determined by the following algorithm when the required pixels I of an operation are scanned simultaneously in parallel. Suppose t_p and t_m are the processor and the external memory critical path delays, respectively.

$$\begin{aligned} &\textit{If } \; t_p / l \cdot k \geq t_m \quad \textit{then} \\ &\qquad t_l = t_p / l \cdot k \qquad (3) \\ &\textit{else} \quad t_l = t_m \end{aligned}$$

Where $l = 2, 3, 4$ ... denote 2, 3, and 4-parallel and t_p/k is the stage critical path delay of a k-stage pipelined processor.

A. 2-parallel pipelined external architecture

Based on the scan method shown in Fig. 3 [5] and DDGs for 5/3 and 9/7 shown in Figs. 1 and 2, respectively, the 2-parallel architecture shown in Fig. 4 is proposed. The dataflow of the architecture is given in Table 1. The architecture is valid for both 5/3 and 9/7 algorithms, since it is developed based on the observation that the DDGs for 5/3 and 9/7 are identical when they are looked at from outside, taking into consideration only inputs and outputs requirements.

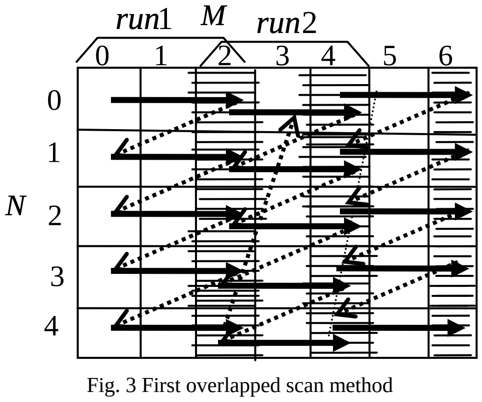

Fig. 3 First overlapped scan method

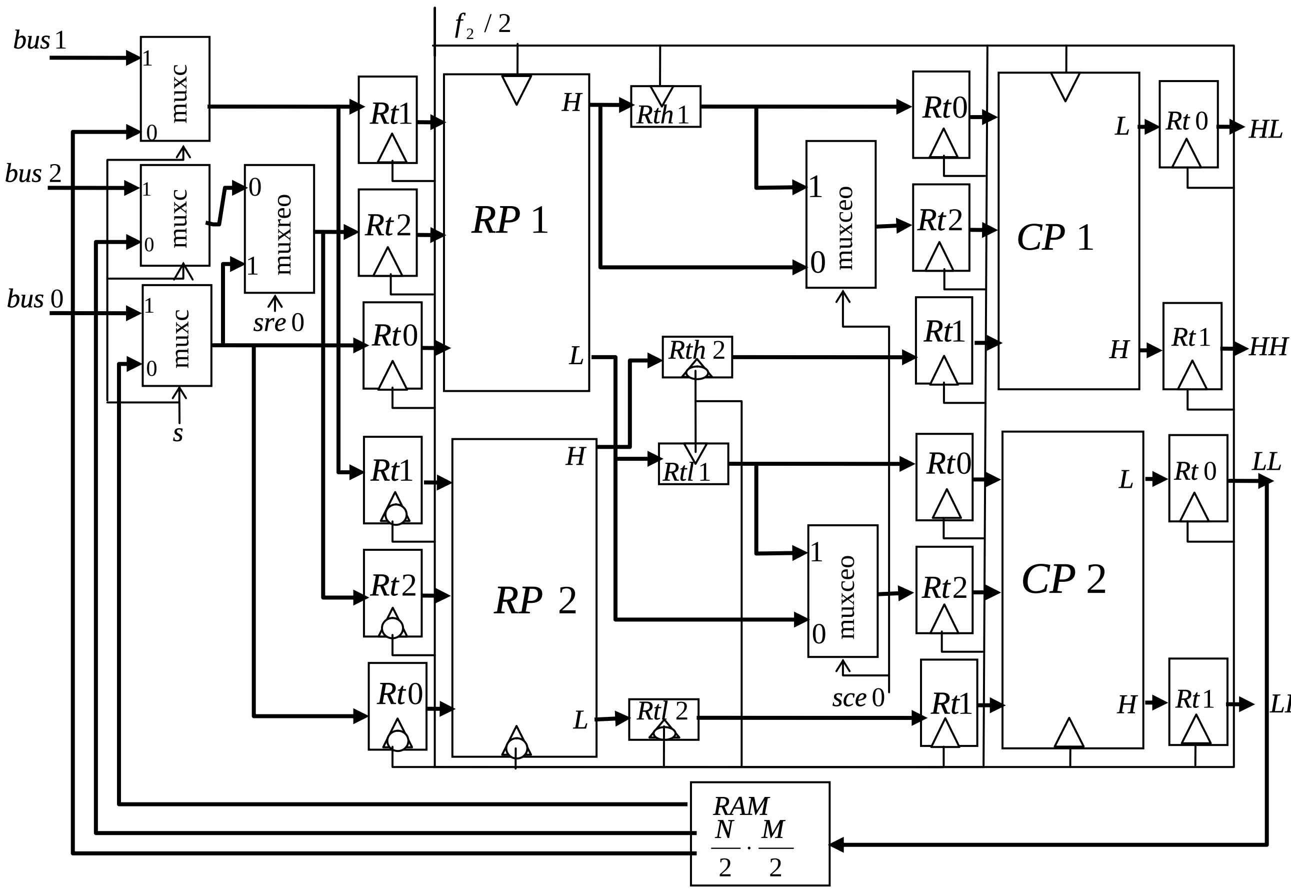

Fig. 4 2-parallel pipelined external architecture

The architecture consists of 2 *k*-stage pipelined row-processors labeled RP1 and RP2 and 2 *k*-stage pipelined column-processors labeled CP1 and CP2. In this architecture, if the external memory is scanned with frequency f_2, then the architecture must operate with frequency $f_2/2$. The buses labeled ***bus0, bus1,*** and ***bus2*** are used for transferring in every clock cycle 3 pixels from external memory to one of the RP latches *Rt0, Rt1,* and *Rt2* to initiate an operation. The RP1's

latches load data every time clock $f_2/2$ makes a positive transition, whereas RP2's latches load data every time a negative transition occurs as indicated in Fig. 4, assuming the first half pulse of the clocks f_2 and $f_2/2$ are low. On the other hand, the column-processors CP1 and CP2 load new data every time clock $f_2/2$ makes a positive transition.

In this architecture, the two RPs decompose an *NxM* image into high (H) and low (L) decompositions, whereas CP1 decomposes H into subbands HH and HL and CP2 decomposes L into subbands LH and LL. CP1 executes high coefficients stored in registers *Rth1* and *Rth2,* while CP2 executes low coefficients stored in registers *Rtl1* and *Rtl2.* That is, in the first run, the first column of H and that of L decomposition will be scheduled for execution by CP1 and CP2, respectively, as shown in Table 1, whereas, in the second run, the second column of both H and L will be executed by CP1 and CP2, respectively and so on.

According to the first overlapped scan method shown in Fig. 3, in any particular time, 3 columns are considered for scanning and in every clock cycle 3 pixels are scanned, one from each column, until end of the columns are reached, say, to complete a run. Then a transition is made to the beginning of the next 3 columns to initiate another run. In the clock cycle where a transition occurs, especially when column length of an image is odd, the external memory should not be scanned, since during that cycle the two CPs each will compute the last low coefficient as required by the DDGs for odd length signals. That is, during that cycle no pixels are loaded into RP2 latches while the control is allowed to return to RP1 by the pulse ending the cycle. This also implies that each run will begin at RP1and the high coefficients generated during a run, which are required in the next run computations, will be stored in the TLB of the RP that generates them. In addition, this scan method requires that each CP to scan(read) coefficients generated by the two RPs column-by-column, which implies that no modifications are needed to the 5/3 and 9/7 datapath architectures developed in [2].

Fig. 5 shows how stage 2 of the pipelined 5/3 and stages 2, 3, and 5 of the pipelined 9/7 developed in [2] should be modified when they are incorporated into the 2-parallel architecture's RPs. The modifications require addition of a TLB of size *N/2* in each stage 2 of the two 5/3 RPs and in each of stages 2, 3, and 5 of the two 9/7 RPs, as shown in Fig. 5. TLBs are necessary, according to the DDGs, to keep *N* high coefficients calculated in step 1 of the 5/3 algorithm and in each of steps 1 and 3 of the 9/7 algorithm including low coefficients calculated in step 2 of the 9/7 algorithm, which are required in the next run computations. In addition, starting

TABLE 1: DATAFLOW FOR THE 2-PARALLEL ARCHITECTURE

CK	RP	RP1 & RP2 Rt0 Rt2 Rt1	Rth Rtl	CP1 Rt0 Rt2 Rt1	CP2 Rt0 Rt2 Rt1	CP1 & CP2 OUTPUTS Rt0 Rt1 Rt0 Rt1
1	1	x0, 0 x0, 2 x0, 1				
2	2	x1, 0 x1, 2 x1, 1				
3	1	x2, 0 x2, 2 x2, 1				
4	2	x3, 0 x3, 2 x3, 1				
5	1	x4, 0 x4, 2 x4, 1				
6	2	x5, 0 x5, 2 x5, 1				
7	1	x6, 0 x6, 2 x6, 1	H0, 0 L0, 0			
8	2	x7, 0 x7, 2 x7, 1	H1, 0 L1, 0			
9	1	x8, 0 x8, 2 x8, 1	H2, 0 L2, 0	H0, 0 H2, 0 H1, 0	L0, 0 L2, 0 L1, 0	
10	2	x9, 0 x9, 2 x9, 1	H3, 0 L3, 0	----------------------	------------------------	
11	1	x10, 0 x10, 2 x10, 1	H4, 0 L4, 0	H2, 0 H4, 0 H3, 0	L2, 0 L4, 0 L3, 0	
12	2	x11, 0 x11, 2 x11, 1	H5, 0 L5, 0	---------------------	------------------------	
13	1	x12, 0 x12, 2 x12, 1	H6, 0 L6, 0	H4, 0 H6, 0 H5, 0	L4, 0 L6, 0 L5, 0	
14	2	x13, 0 x13, 2 x13, 1	H7, 0 L7, 0	---------------------	------------------------	
15	1	x14, 0 x14, 2 x14, 1	H8, 0 L8, 0	H6, 0 H8, 0 H7, 0	L6, 0 L8, 0 L7, 0	HH0, 0 HL0, 0 LH0, 0 LL0, 0
16	2	x15, 0 x15, 2 x15, 1	H9, 0 L9, 0	----------------------	----------------------	------------------------------------
17	1	x16, 0 x16, 2 x16, 1	H10, 0 L10, 0	H8, 0 H10, 0 H9, 0	L8, 0 L10, 0 L9, 0	HH1, 0 HL1, 0 LH1, 0 LL1, 0
18	2	x17, 0 x17, 2 x17, 1	H11, 0 L11, 0	----------------------	----------------------	-------------------------------------
19	1	x18, 0 x18, 2 x18, 1	H12, 0 L12, 0	H10, 0 H12, 0 H11, 0	L10, 0 L12, 0 L11, 0	HH2, 0 HL2, 0 LH2, 0 LL2, 0

from the second run, it is required that the TLB must be read and written in the same clock cycle. Therefore, signal $\overline{R}/W$ (read/write) is connected to the clock $f_2/2$ in Fig. 5 so that the TLB can be read in the first half cycle and written in the second half. The data read in the first half cycle, for example, from TLB1, is stored in register Rd1 by the negative transition of the clock $f_2/2$. Then the positive transition of the clock loads it into the latch of the next stage.

The register labeled TLBAR (TLB address register) generates addresses for the TLB. Initially, register TLBAR is cleared to zero by asserting signal INCAR low to point at the first location in the TLB. Then to address the next location after each read and write, register TLBAR is incremented by one by asserting INCAR high.

The DDGs for even length signals show that in the last high and low coefficients calculations, only the last two pixels in a row, r, at locations *X(r, M-2)* and *X(r, M-1)* are read from external memory. In addition, the extension part of the DDGs for even length requires the pixel located at *X(r, M-2)* to be considered as the first and the third inputs. This pixel must be passed to the RP2 with the second input pixel from location *X(r, M-1),* to compute the last high and low coefficients in row r. Thus, the multiplexer labeled muxre0, which is an extension multiplexer, passes in all cases data coming through bus2, except when the row length *(M)* of an image is even and only in the calculations of the last high and low coefficients in a row *r*, the pixel of location *X (r, M-2),* which will be read into bus0, must be allowed to pass through muxre0 and then loaded into *Rt2* as well as *Rt0*. The two multiplexers labeled muxce0, attached to CPs, are also extension multiplexers and operate similar to muxre0 when DWT is applied column-wise by CPs. The three multiplexers, labeled *muxc* allow either the

external memory or the LL-RAM data to be passed to the RPs latches *Rt0, Rt1,* and *Rt2*.

On the other hand, when the row length of an image is odd, according to the DDGs for odd length signals, to calculate the last low coefficient, only one pixel, the last one at location *X(r, M-1),* should be passed to the RP1.

The dataflow of the architecture shown in Table 1 is for 5/3. This dataflow table is identical to 9/7 dataflow except in the first run where 9/7 RPs, according to the 9/7 DDGs, will not yield any output coefficients by processing the first 3 pixels of each row. The 9/7 RPs, in the first run, will be able to compute only two coefficients labeled $Y''(1)$ and $Y''(0)$ in the DDGs for each row and these coefficients can be stored in temporary line buffers (TLBs) so that they can be used in the next run computations.

In the following, the dataflow of the architecture accompanied by Table 1 will be described. For convenience, assume the processors are pipelined to *(k = 3)*-stages. In the first clock cycle, reference to clock f_2, 3 pixels are scanned from the external memory and are loaded into the RP1's latches by the positive transition of clock $f_2/2$ to initiate the first operation. The second cycle scans another 3 pixels from external memory and loads them into the RP2's latches by the negative transition of clock $f_2/2$ to initiate the second operation. This process is repeated until the whole image pixels are scanned as shown in Table 1.

In cycle 7, the first outputs of the RP1, labeled H0, 0 and L0, 0 in the table, are loaded into the latches labeled *Rth*1 and

*Rtl*1, respectively, by the positive transition of $f_2/2$. In cycle 8, the RP2 produces its first outputs H1, 0 and L1, 0 which are loaded into its output latches *Rth2* and *Rtl2,* respectively, by the negative transition of $f_2/2$. In cycle 9, the RP1 produces

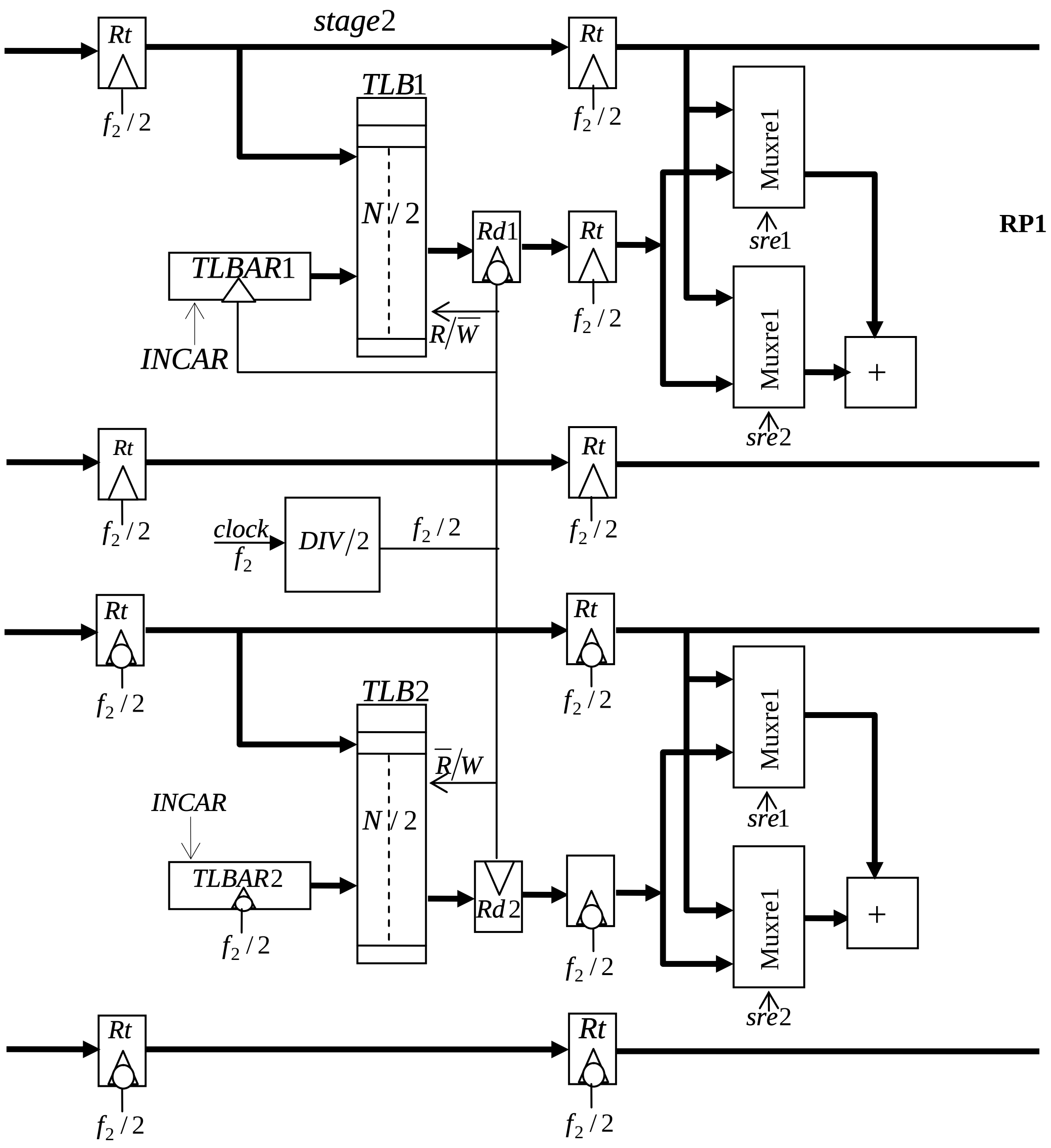

Fig. 5.Modified stage 2 of the RP of the 2-parallel

its second outputs H2, 0 and L2, 0. Then by the positive transition of clock $f_2/2$, the high coefficients H0, 0, H2, 0, and H1, 0, and the low coefficients L0, 0, L2, 0, and L1, 0 are loaded into the latches of the CP1 and CP2 labeled *Rt0, Rt2,* and *Rt1*, respectively. The same positive transition also stores H2, 0 and L2, 0 coefficients in the output latches of the RP1 labeled Rth1 and Rtl1, respectively, since they are also required in the next high and low coefficients calculations. Note that according to the DDGs, each operation initiated by either RP or CP requires 3 inputs.

In cycle 15, the RP1 yields its fifth outputs H8, 0 and L8, 0, while CP1 and CP2 both yield their first outputs HH0, 0, HL0, 0, LH0, 0 and LL0, 0. Then by the positive transition of $f_2/2$, the first outputs of CP1 HH0, 0 and HL0, 0 and the first outputs of CP2 LH0, 0 and LL0, 0 are loaded into the output latches *Rt1* and *Rt0* of the CP1 and the CP2, respectively. According to Table 1, every other clock cycle or every clock cycle with respect to clock $f_2/2$ two pairs of output coefficients will be generated by both CP1 and CP2.

A. 4-parallel pipelined external architecture

The 4-parallel architecture is shown in Fig. 6 and its dataflow is given in Table 2. This architecture closely resembles the 2-parallel architecture. The main difference is that the 2-parallel architecture consists of two pipelined processors, whereas the 4-parallel consist of 4 pipelined processors. Each pipelined processor contains one RP and one CP. The architecture scans the external memory with frequency f_4 and operates with frequency $f_4/2$. The clock frequency f_4 can be obtained from Eq (3) as.

$$f_4 = 4k/t_p \tag{4}$$

Two waveforms of the frequency $f_4/4$ labeled f_{4a} and f_{4b} which can be generated from f_4 are shown in Fig. 7. Note that

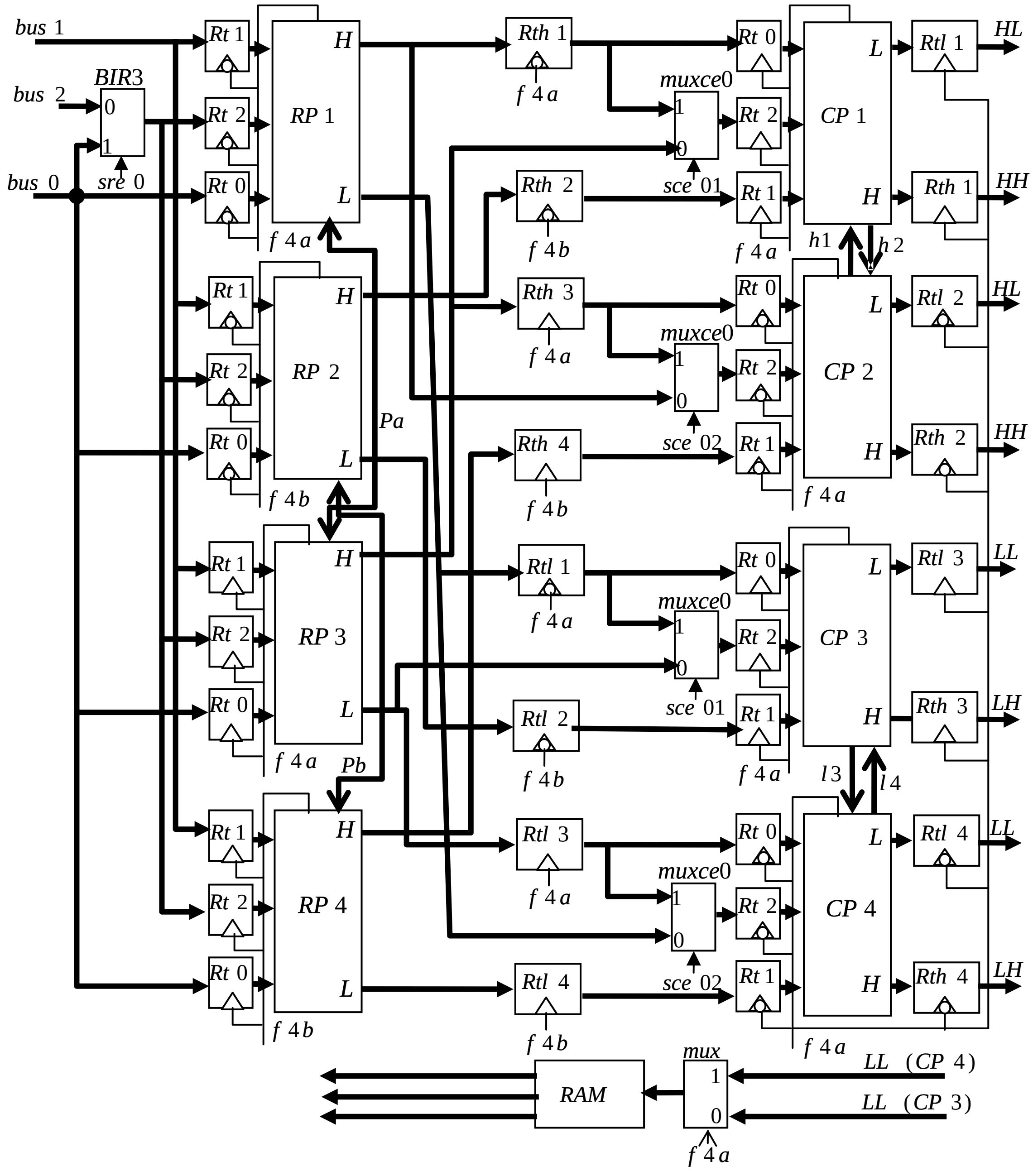

Fig. 6 4-parallel pipelined external architecture

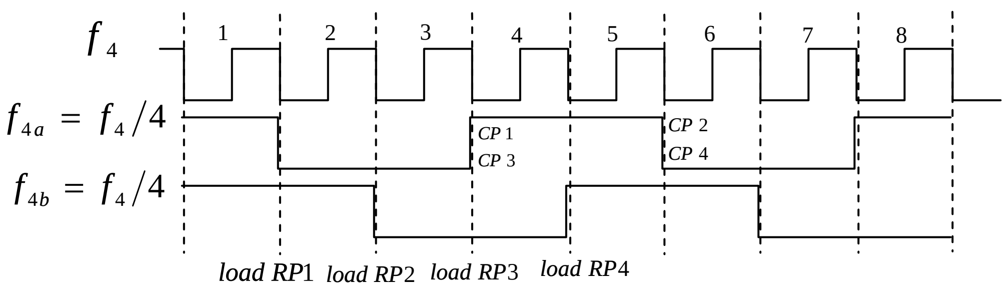

Fig. 7 Waveforms of the 3 clocks used in 4-parallel

TABLE II: DATAFLOW FOR 4-PARALLEL ARCHITECTURE

ck	RP	RP's input latches Rt0 Rt2 Rt1	RP's output latches Rth Rtl	CP1 & CP3 input latches Rt0 Rt2 Rt1 Rt0 Rt2 Rt1
1	1	x 0, 0 x 0, 2 x 0, 1		
2	2	x 1, 0 x1, 2 x 1, 1		
3	3	x 2, 0 x 2, 2 x 2, 1		

4	4	x 3, 0 x 3, 2 x 3, 1		
5	1	x 4, 0 x 4, 2 x 4, 1		
6	2	x 5, 0 x 5, 2 x 5, 1		
7	3	x 6, 0 x 6, 2 x 6, 1		
8	4	x 7, 0 x 7, 2 x 7, 1		
9	1	x 8, 0 x 8, 2 x 8, 1		
10	2	x 9, 0 x 9, 2 x 9, 1		
11	3	x 10, 0 x 10, 2 x 10, 1		
12	4	x 11, 0 x 11, 2 x 11, 1		
13	1	x 12, 0 x 12, 2 x 12, 1	H0, 0 L0, 0	
14	2	x 13, 0 x 13, 2 x 13, 1	H1, 0 L1, 0	
15	3	x 14, 0 x 14, 2 x 14, 1	H2, 0 L2, 0	H0, 0 H2, 0 H1, 0 L0, 0 L2, 0 L1, 0
16	4	x 15, 0 x 15, 2 x 15, 1	H3, 0 L3, 0	--
17	1	x 16, 0 x 16, 2 x 16, 1	H4, 0 L4, 0	--
18	2	x 17, 0 x 17, 2 x 17, 1	H5, 0 L5, 0	--
19	3	x 18, 0 x 18, 2 x 18, 1	H6, 0 L6, 0	H4, 0 H60 H5, 0 L4, 0 L6, 0 L5, 0
20	4	x 19, 0 x 19, 2 x 19, 1	H7, 0 L7, 0	--
21	1	x 20, 0 x 20, 2 x 20, 1	H8, 0 L8, 0	--
22	2	x 21, 0 x 21, 2 x 21, 1	H9, 0 L9, 0	--
23	3	x 22, 0 x 22, 2 x 22, 1	H10, 0 L10, 0	H8, 0 H10, 0 H9, 0 L8, 0 L10, 0 L9, 0
24	4	x 23, 0 x 23, 2 x 23, 1	H11, 0 L11, 0	--
25	1	x 24, 0 x 24, 2 x 24, 1	H12, 0 L12, 0	--
26	2	x 25, 0 x 25, 2 x 25, 1	H13, 0 L13, 0	--
27	3	x 26, 0 x 26, 2 x 26, 1	H14, 0 L14, 0	H12, 0 H14, 0 H13, 0 L12, 0 L14, 0 L13, 0
28	4	x 27, 0 x 27, 2 x 27, 1	H15, 0 L15, 0	--
29	1	x 28, 0 x 28, 2 x 28, 1	H16, 0 L16, 0	--

CK	CP2 &CP4 input latches Rt0 Rt2 Rt1 Rt0 Rt2 Rt1	CP1 & CP3 output latches Rth1 Rtl1 Rth3 Rtl3	CP2 & CP4 output latches Rth2 Rtl2 Rth4 Rtl4
17	H2, 0 H4, 0 H3, 0 L2, 0 L4, 0 L3, 0		
18	---		
19	---		
20	---		
21	H6, 0 H8, 0 H7, 0 L6, 0 L8, 0 L7, 0		
22	---		
23	---		
24	---		
25	H10, 0 H12, 0 H11, 0 L10, 0 L12, 0 L11, 0		
26	---		
27	---	HH0, 0 HL0, 0 LH0, 0 LL0, 0	------------------------------------
28	---	------------------------------------	------------------------------------
29	H14, 0 H16, 0 H15, 0 L14, 0 L16, 0 L15, 0	------------------------------------	HH1, 0 HL1, 0 LH1, 0 LL1, 0

when degree of parallelism increases from 2 to 4 e.g., the scanning frequency f_l also increases, while the architecture frequency of operation f_l/l, which is the reciprocal of the stage critical path delay (t_p/k) of the pipelined processors, remains the same.

In the architecture, RP1 and RP3, and their associate latches employ the clock f_{4a}, whereas RP2 and RP4 and their associate latches employ the clock f_{4b} as shown in Fig. 6

In every clock cycle, reference to clock f_4, three pixels are scanned from external memory and are loaded into the latches of one of the RPs. First, RP1 latches are loaded followed by RP2 latches then RP3 latches followed by RP4 latches. This process then repeats. When the scanning process return to RP1 to initiate another operation, the RP1 should have completed its current operation in the time specified by t_p/k, and should be ready to accept pixels of the next operation. As indicated in the architecture, RP1 latches will be loaded with new data every time clock f_{4a} makes a negative transition, while RP3 latches will be loaded at the positive transition. On the other hand, RP2 and RP4 latches will be loaded at the negative and positive transition of the clock f_{4b} respectively.

In the 2-parallel architecture, each new run begins at RP1and high coefficients generated during a run, which are require in the next run, are stored in the TLB of the RP that generated them. In the 4-parallel, the strategy adopted allows each new run to start its computations in the RP that immediately comes after the RP where the previous run ends. Which implies a run can begin at any RP and that makes the decision where to store the high coefficients needed in the next run somewhat difficult. Thus, for this strategy to be effective, we have come up with a simple scheme that allows such decision to be made.

The scheme, which can be figured out by scheduling several short runs, is summarized as follows. Taking into consideration the fact that in the clock cycle where a transition from a run to the next is made, the external memory is not scanned and thus, no pixels are loaded into RP latches, when column length of an image is odd, then the decision, where to store each high coefficient calculated in the previous run that is needed in the calculation of a low coefficient in the
next run, can be made by examining the two least significant bits of *N*. case one; if the two least significant bits of *N* are 00 or 11 then the high coefficients should be stored in the TLBs of the RPs that generate them. Case two: if the two least significant bits of *N* are 01 or 10, then the high coefficients of RP1 should be stored in the TLB of RP3 and vice versa, and the high coefficients of RP2 should be stored in the TLB of RP4 and vice versa. Symbolically, case two can be written as

$$\begin{aligned} RP1 &\xleftrightarrow{Pa} RP3 \\ RP2 &\xleftrightarrow{Pb} RP4 \end{aligned} \tag{5}$$

Therefore, the paths labeled *Pa* and *Pb* are added in Fig. 6.

The above scheme only affects stage 2 of the four 5/3 RPs and stages 2, 3, and 5 of the four 9/7 RPs developed in [3] and it can be implemented as shown in Fig. 8. The strategy is very simple to control, since it only adds one control signal *(zs)* that is used to control the operation of the four multiplexers labeled *mux1, mux2, mux3*, and *mux4.* Signal *zs* can be generated by use of a simple 2-input XNOR gate with its two inputs connected to the two least significant bits of *N*. Thus, if the input to the XNOR are either 00 or 11 (case one), *zs* is asserted high to pass the high coefficient stored in each Rt of stage 2 which is generated in stage 1 of the same RP. Otherwise (case two) it is asserted low to pass the high coefficient stored in each register *BIR* that have been generated by one of the RP. Note that signal *zs* will only have one value during each level of decomposition. For example, during the whole period of the first level decomposition, *zs* may be equal to 1 or 0, but not both. The advantage of this arrangement is that all operations, in stage 2 of the four RPs in Fig. 8, such as read and write into TLBs are controlled internally by the two clock signals f_{4a} and f_{4b} with no control unit engagement once the 3 clocks are synchronized. Only the control signal values for signal *INCAR* will be generated by a control unit.

Now, let’s move to the CPs side to see how this part of the architecture works. The 4 CPs run by the clock labeled f_{4a}. Both CP1 and CP3 load new data every time clock f_{4a} makes a positive transition, whereas, both CP2 and CP4 load new data every time clock f_{4a} makes a negative transition. However, both CP1 and CP2 execute high coefficients stored in *Rth1, Rth2, Rth3,* and *Rth4*, whereas CP3 and CP4 execute low coefficients stored in *Rtl1, Rtl2, Rtl3,* and *Rtl4.*

Suppose, in clock cycle *n* (cycle 13 in Table 2), the first two output coefficients H0, 0 and L0, 0 generated by RP1 are loaded into its output latches labeled *Rth*1and *Rtl*1, respectively. In cycle *n*+1, RP2 loads coefficients H1, 0 and L1, 0 into *Rth2* and *Rtl2*, respectively. In cycle *n*+2, the high coefficients in *Rth*1 and *Rth2* along with coefficient H2, 0 generated by RP3 during the cycle and the low coefficients in *Rtl*1 and *Rtl*2 along with coefficient L2, 0 are loaded into CP1 and CP3 input latches, respectively, while coefficients H2, 0 and L2, 0 generated by RP3 during the cycle are loaded as well into *Rth3* and *Rtl3*, respectively. Cycle *n* +3, loads coefficients H3, 0 and L3, 0 generated by RP4 into *Rth4* and *Rtl4*, respectively. In cycle *n+4*, the high coefficients in *Rth*3 and *Rth*4 along with coefficient H4, 0 generated by RP1 during the cycle and the low coefficients in *Rtl*3 and *Rtl*4 along with coefficient L4, 0 are loaded into CP2 and CP4 input latches, respectively, while coefficients H4, 0 and L4, 0 generated by RP1 during the cycle are loaded into *Rth*1and *Rtl*1, respectively, as well. Thi process is repeated until the level decomposition completes.

In cycle *n*+14, CP1 and CP3 yield their first 4 output coefficients HH0, 0, HL0, 0 and LH0, 0, LL0, 0, respectively. No output is generated in cycle *n*+15. In cycle *n*+16, CP2 and CP4 yield their first 4 outputs HH1, 0, HL1, 0 and LH1, 0, LL1, 0, respectively. Thus, every other clock, reference to clock f_4, two pairs of output coefficients will be generated as shown in Table 2.

According to Table 2, in each run, CP1 and CP2 together and similarly CP3 and CP4 are required to execute one column of H and one column L decomposition, respectively, which would require interactions between two CPs. That is, according to the DDGs for 5/3 and 9/7, a coefficient calculated in the first stage of a CP is also required in the calculation of the coefficient that takes place in the second stage of the other CP and vise versa. However, this would required CP1 and CP2 similarly CP3 and CP4 datapaths to be modified as shown in Fig. 9 for CP1 and CP2, which is identical to CP and CP4. Fig. 9 shows that passing coefficients occur between stages 2 of the two CPs, in the case of the 5/3 and between stages 2, between stages 3, and between stages 5 of the two CPs, in the case of the 9/7. Note that the modified stages 5 of the two 9/7 CPs are identical to stages 2 of CP1 and CP2 shown in Fig. 9.

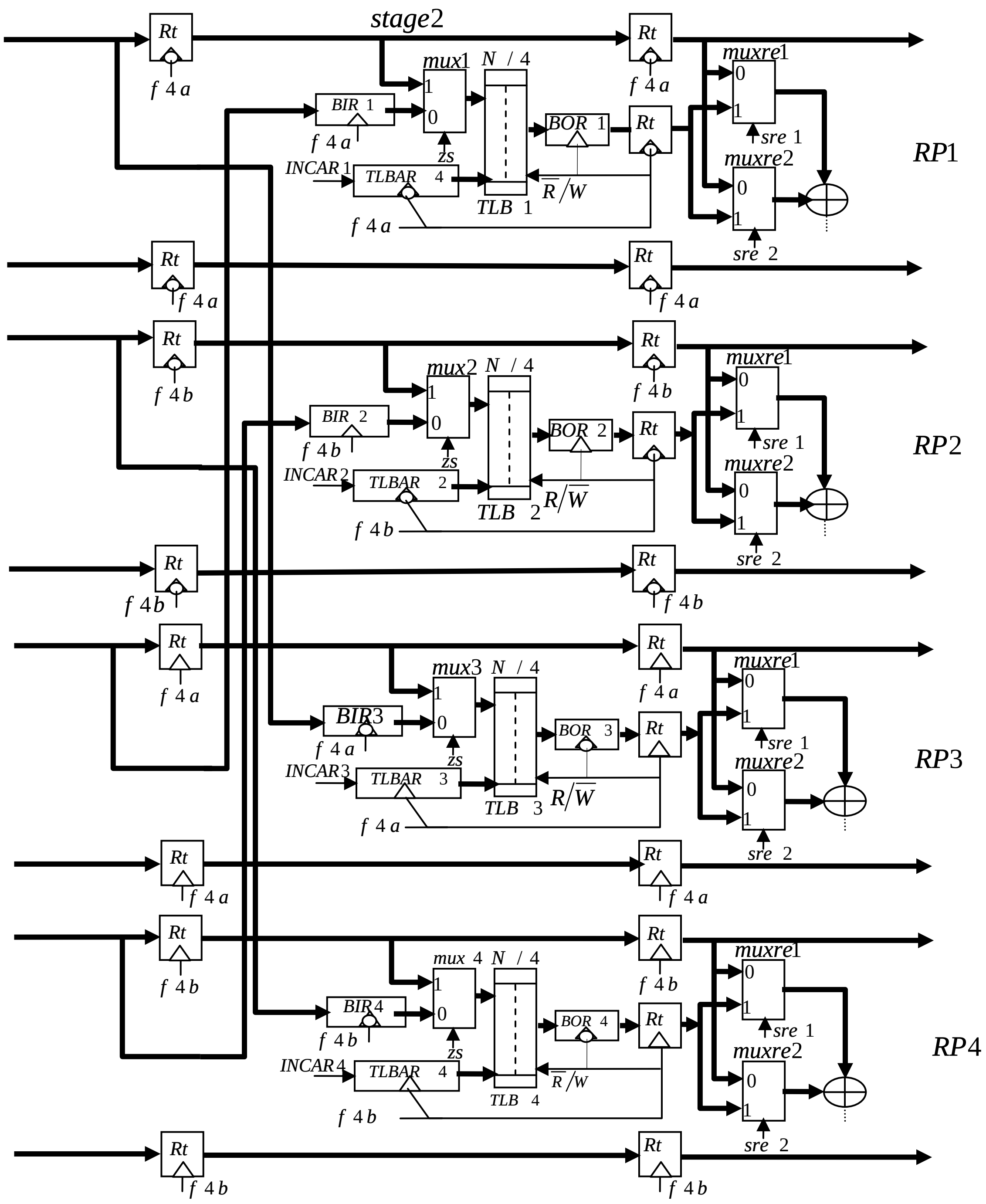

Fig. 8 Modified stage 2 of the RPs datapath architecture

In a control design it would be necessary to determine the clock cycle *(C*1) where the first input data are loaded into the CPs latches and the clock cycle (*C2*) where the first output coefficients are loaded into the CPs output latches. The following two equations can be used to determine *C*1 and *C2.*

$$C1 = l \cdot k_r + 2 + 1 \qquad (6)$$

$$C2 = C1 + l \cdot k_c \qquad (7)$$

Where l = 2, 3, 4… denote 2-, 3-, 4-parallel (degree of parallelism) and so on. K_r and k_c are the number of pipeline stages in a RP and a CP, respectively.

I. EVALUATION OF ARCHITECTURES

To evaluate the performances of the two proposed parallel architectures, in terms of speedup, efficiency, and power consumption consider the following. In the single pipelined processor architecture based on the first overlapped scan method [2], the total time *T*1 required to yield *n* pairs of output for j-level decomposition of an *NxM* image is given by

$$T1 = [r1 + 3(n-1)]t_1$$

$$T1 = [r1 + 3(n-1)]t_p / 3k \qquad (8)$$

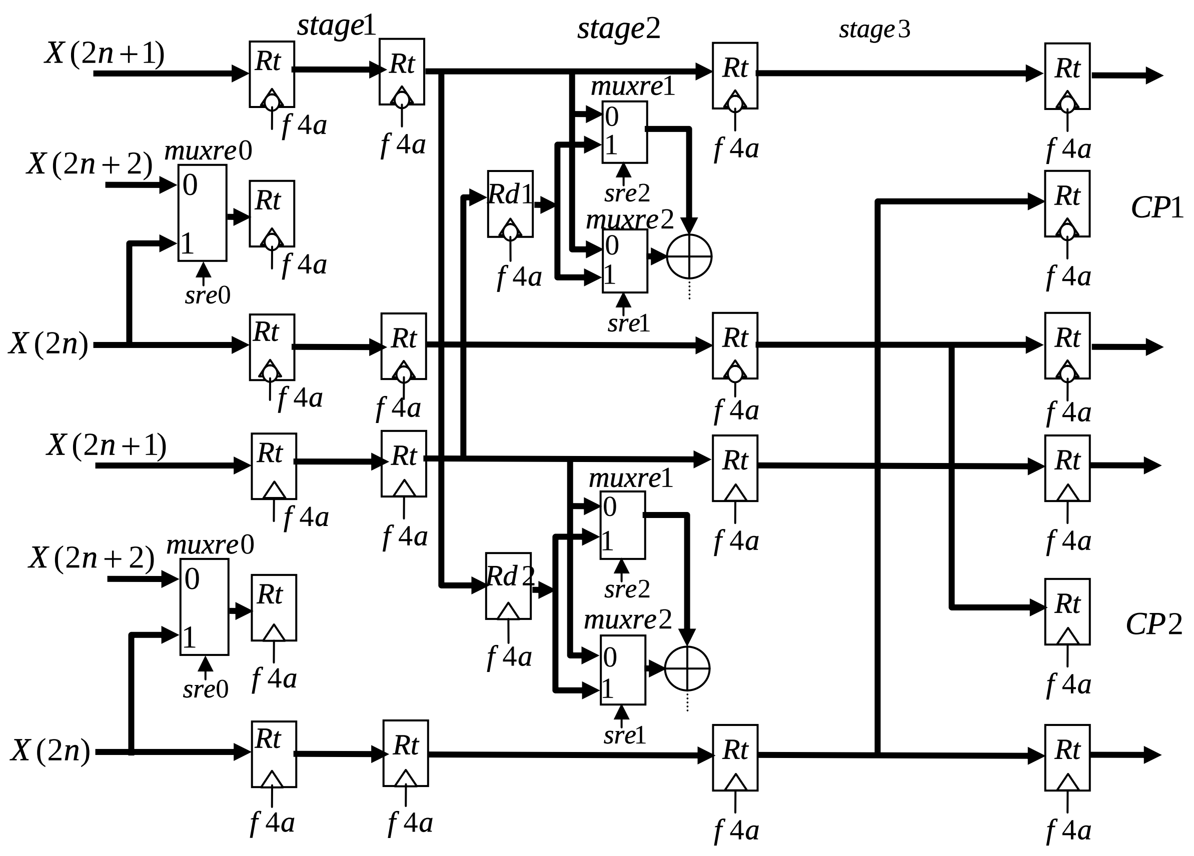

Fig. 9 Modified CP1 and CP2 datapaths for 4-parallel architecture

Where $r1$ is number of clock cycles needed to yield the first pair of output coefficients by CP.

On the other hand, the total time, *T2*, required to yield *n* pairs of output coefficients for j-level decomposition of an *NxM* image on the 2-parallel pipelined architecture shown in Fig. 4, can be estimated using Table 1 as

$$T2 = \frac{[r2 + 2(n-1)]t_2}{2}$$

From (3), $t_2 = t_p/2k$

Therefore, $$T2 = \frac{[r2 + 2(n-1)]t_p/2k}{2} \quad (9)$$

The speedup factor (*S2*) is then given by

$$S2 = \frac{T1}{T2} = \frac{[r1 + 3(n-1)]t_p/3k}{[r2 + 2(n-1)]t_p/4k}$$

For large *n*, the above equation reduces to

$$S2 = \frac{3(n-1)t_p/3k}{2(n-1)t_p/4k} = 2 \quad (10)$$

The efficiency $E_2 = S_2/2 = 1$ (11)

Eqs (10) and (11) indicate that the 2-parallel architecture is 2 times faster than the single pipelined architecture and efficiency 1, respectively.

Similarly, the total time (*T4*) require to yield *n* pairs of output for j-level decomposition of an *NxM* image on the 4-parallel pipelined architecture can be written as

$$T4 = \frac{[r4 + 2(n-1)]t_4}{2}$$

From (3) $t_4 = t_p/4k$

Thus, $$T4 = \frac{[r4 + 2(n-1)]t_p/4k}{2} \quad (12)$$

The speedup factor (*S4*) is then given by

$$S4 = \frac{T1}{T4} = \frac{[r1 + 3(n-1)]t_p/3k}{[r4 + 2(n-1)]t_p/8k}$$

For large *n*, the above equation reduces to

$$S4 = \frac{24(n-1)}{6(n-1)} = 4 \quad (13)$$

The efficiency $E_4 = S_4/4 = 1$ (14)

Equations (13) and (14) imply that the 4-parallel architecture is 4 times faster than the single pipelined architecture and the efficiency is 1, respectively.

On the other hand, the power consumption of *l*-parallel pipelined architecture as compared with the single pipelined architecture can be obtained as follows. Let P_1 and P_l denote the power consumption of the single and *l*-parallel architectures without the external memory, and P_{m1} and P_{ml} denote the power consumption of the external memory for the single and *l*-parallel architectures, respectively. If the power consumption of VLSI architecture can be estimated as

$$P = C_{total} \cdot V_0^2 \cdot f$$

where C_{total} denotes the total capacitance of the architecture, V_0 is the supply voltage, and f is the clock frequency, then,

$$P_1 = C_{total} \cdot V_0^2 \cdot f_1/3 \ , \ \ P_l = l \cdot C_{total} \cdot V_0^2 \cdot f_l/l$$

and

$$\frac{P_l}{P_1} = \frac{l \cdot C_{total} \cdot V_0^2 \cdot f_l / l}{C_{total} \cdot V_0^2 \cdot f_1 / 3} = \frac{3 f_l}{f_1} = 3\left(\frac{l \cdot k}{t_p}\right) \Big/ 3k/t_p = l \qquad (15)$$

Whereas, P_{m1} and P_{ml} can be estimated as
$P_{m1} = C_{total}^{m} \cdot V_0^2 \cdot f_1$, $P_{ml} = 3 \cdot C_{total}^{m} \cdot V_0^2 \cdot f_l$, and

$$\frac{P_{ml}}{P_{m1}} = \frac{3 \cdot f_l}{f_1} = \frac{3 \cdot l \cdot k/t_p}{3k/t_p} = l \qquad (16)$$

Where C_{total}^{m} is the total capacitance of the external memory.

From the above evaluations, it can be concluded that as degree of parallelism increases the speedup and the power consumption of the architecture, without external memory, and the power consumption of the external memory increase by a factor of *l*, as compared with single pipelined architecture.

I. COMPARISON RESULTS

A comparison of the proposed architectures with most recent architectures in the literature is illustrated in Table 3. Flipping structure [6] introduces a new method to shorten the critical path delay of the lifting-based architecture to one multiplier delay but requires a total line buffer of size 11N [7], [8], which is a very expensive memory component. In [7], a more efficient structure than flipping is present which reduces the total line buffer size to 6.5N.

On the other hand, in [8], by reordering the lifting-based DWT of the 9/7 filter; the critical path of the pipelined architecture has been reduced to one multiplier delay. But, this architecture also requires a total line buffer of size 5.5N. The architecture proposed in [9], achieves critical path of one multiplier delay using very large number of pipeline registers, 52. In addition, it requires a total line buffer of size 6N. The architecture proposed in [10], like the proposed 2-parallel architecture, achieves a speedup factor of 2 but, requires a total line buffer of size 5.5 N, while the two proposed parallel architectures each require only a total line buffer of size *3N.*

TABLE III: COMPARISON OF SEVERAL 9/7 2-DWT ARCHITECTURES

Architecture	**Mult**	**Adders**	**Line buffer**	**Computin g time**	**Critical path**
Flipping [6]	**10**	**16**	**11N**	**N/A**	**Tm**
PLSA [7]	**12**	**16**	**6.5N**	**N/A**	**Tm**
Bing [8]	**6**	**8**	**5.5N**	$2(1-4_{-j})N_2$	**Tm**
Lan [9]	**12**	**12**	**6N**	$2(1-4_{-j})N_2$	**Tm**
Overlapped [5]	**10**	**16**	**3N**	$2(1-4_{-j})N_2$	Tm + 2Ta
Cheng [10]	**18**	**32**	**5.5N**	$(1-4_{-j})N_2$	**N/A**
Proposed 2-parallel	**18**	**32**	**3N**	$(1-4_{-j})N_2$	Tm + 2Ta
Proposed 4-parallel	**36**	**64**	**3N**	$1/2\ (1-4_{-j})N_2$	Tm + 2Ta

Tm: multiplier delay Ta: adder delay

I. CONCLUSIONS

In this paper, 2-parallel and 4-parallel pipelined architectures for 2-D DWT are proposed. The two proposed architectures achieve speedup factors of 2 and 4, respectively, as compared with single pipelined architecture. The scan method adopted not only reduces the internal memory between RPs and CPs to a few registers, but allows CPs to work in parallel with RPs earlier during the computation. In addition, the comparison results show that the proposed parallel architectures only require a total temporary line buffer (TLB) of size N and 3N in 5/3 and 9/7, respectively, while other architectures require more line buffers, which are very expensive memory components. That implies, the proposed architectures would occupy less silicon areas and would cost less. Therefore, the proposed architectures could be very efficient alternative in 2-D DWT applications requiring very high-speed and low power. Furthermore, the proposed architectures are simple to control and their control algorithms can be immediately developed.

REFERENCES

[1] David S. and Michael W. “JPEG 2000 image compression fundamentals, standards and practice, ” Kluwer Academic pulishers, 2002.

[2] Ibrahim Saeed, H. Agustiawan., “Lifting-based VLSI architectures for 2-dimensional discrete wavelet transform for effective image Compression, ” Proceedings of the International MultiConference of

Engineers and Computer Scientists 2008 Vol. 1 IMECS'08, Hong Kong, PP. 339-347, Newswood Limited, 2008.

[3] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting schemes, " J. Fourier Analysis and Application Vol. 4, No. 3, 1998, PP. 247-269.

[4] G. Dillin, B. Georis, J-D. Legant, O. Cantineau, "Combined Line-based Architecture for the 5-3 and 9-7 Wavelet Transform of JPEG2000, "IEEE Trans. on circuits and systems for video tech., Vol. 13, No. 9, Sep. 2003, PP. 944-950.

[5] I. Saeed and Herman Agustiawan, "high-speed and power Efficient lifting-based VLSI architectures for two-dimensional discrete Wavelet transform, " proceedings of the IEEE Second Asia International Conference on Modelling and Simulation, AMS 2008, PP. 998-1005.

[6] C.-T. Huang, P.-C. Tseng, L.-G. Chen, "Flipping structure: Anefficient architectures for 1-D and 2-D lifting-Based wavelet transform, " IEEE Tran. Signal Proc., Vol. 52, No. 4, April 2004, PP. 1080 -1089.

[7] C. Yi, J. Wen, J. Liu, "A note on Flipping structure: an efficient VLSI architecture for lifting-based discrete wavelet transform, " IEEE Transaction on signal proc. Vol. 54, No. 5, May 2006, PP. 1910 – 1916.

[8] B-F. Wu, C-F. Lin, "A high-Performance and Memory-Efficient Pipeline Architecture for the 5/3 and 9/7 Discrete Wavelet Transform of PEG2000 Codec, " IEEE Trans. on Circuits & Sys. for Video Technology, Vol. 15, No. 12, December 2005, PP. 1615 – 1628.

[9] X. Lan, N. Zheng, " Low-Power and High-Speed VLSI Architecture for Lifting-Based Forward and Inverse Wavelet Transform, " IEEE trans. on consumer electronics, Vol. 51, No. 2, May 2005, PP. 379 – 385.

[10] C-Y. Xiong, J-W. Tian, J. Liu, "Efficient high-speed/low-power line based architecture for two-dimensional discrete wavelet transforms using lifting scheme, " IEEE Trans. on Circuits & sys. For Video Tech. Vol.16, No. 2, February 2006, PP. 309-316.