Parallel Form of the Pipelined Intermediate Architecture for Two-dimensional Discrete Wavelet Transform

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

Abstract—**A lifting-based VLSI architecture for two-dimensional discrete wavelet transform (2-D DWT) for 5/3 and 9/7 algorithms, called, pipelined intermediate architecture was proposed by *Ibrahim et al.*, which aim at reducing power consumption of the overlapped external memory access without using the expensive line-buffer. In this paper, we explore parallelism in order to best meet real-time applications of 2-D DWT with demanding requirements in terms of speed, throughput, and power consumption. Therefore, 2-parallel and 3-parallel form of the single pipelined intermediate architecture are proposed. The 2-parallel and 3-parallel pipelined intermediate architectures achieve speedup factors of 2 and 3, respectively, as compared with single pipelined intermediate architecture proposed by *Ibrahim et al.***

***Index Terms*—Parallel intermediate architecture, discrete wavelet transform, lifting scheme, and VLSI.**

I. INTRODUCTION

The rational behind developing intermediate architecture in [3] was to reduce the excess power consumption of the external memory, due to scanning overlapped areas, to somewhat between the architecture based on the first overlapped scan method and that based on the nonoverlapped scan method [10]. The intermediate architecture reduces power consumption by 22.2 % as compared with the architecture based on the first overlapped scan method proposed in [10]. This reduction in power consumption is achieved without the expensive line-buffer (LB) used in the nonoverlapped scan architecture proposed in [10]. The nonoverlapped architecture reduces the external memory access to minimum, *NM* cycles and hence the power consumption. Intermediate architectures are based on the generalization of the overlapped scan method proposed in [3].

In this paper, the single pipelined intermediate architecture is extended to 2-parallel and 3-parallel architectures for 5/3 and the 9/7 algorithms. The proposed 2-parallel and 3-perallel pipelined intermediate architectures achieves speedup factors of 2 and 3, respectively, as compared with single pipelined intermediate architecture. The two proposed parallel architectures are intended for used in real-time applications of 2-D DWT, where very high-speed and low-power are required. The advantage of the proposed parallel architectures is that the total temporary line buffer (TLB) does not increase from that of the single pipelined intermediate architecture when degree of parallelism is increased. The two proposed parallel architectures are based on lifting scheme, which facilitates high speed and efficient implementation of wavelet transforms [1], [2].

This paper is organized as follows. In section II, the lossless 5/3 and the lossy 9/7 algorithms are stated and the data dependency graphs (DDGs) for both algorithms are given. In section III, The two proposed parallel architectures are presented. Evaluation and scale multipliers reduction are given in sections IV and V, respectively. Comparisons are discussed in sections IV. Conclusions are given in section V.

The authors are with the Electrical and Electronic Engineering Department, Universiti Teknologi PETRONAS, Perak, Tronoh, Malaysia (emails: kokois12@hotmail.com, herman_agustiawan@petronas.com.my).

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]:

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

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

The data dependency graphs (DDGs) for 5/3 and 9/7 derived from the algorithms are shown in Figs. 1 and 2, respectively. 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 input. The boundary treatment is only applied at the beginning and ending of each row and column in an *NxM* image [6] to prevent distortion from appearing at the image boundaries. The nodes circled with the same numbers, are considered redundant computations, which will be computed once and used thereafter.

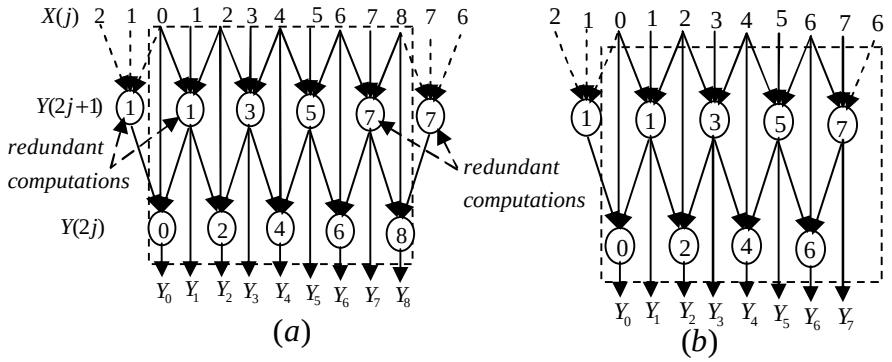


Fig. 1. 5/3 algorithm's DDGs for (a) odd and (b) even length signals

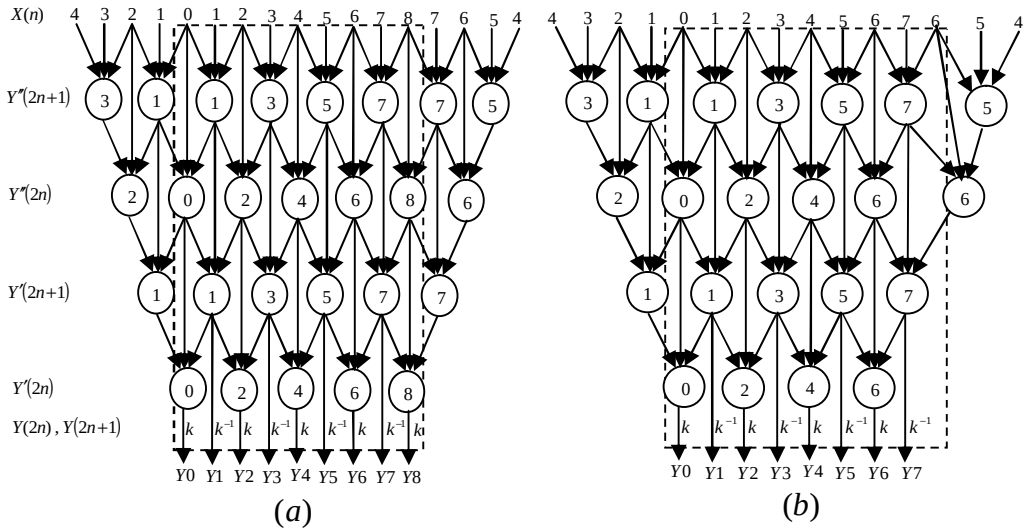


Fig. 2. 9/7 algorithm's DDG for odd (a) and even (b) length signals

III. PROPOSED PARALLEL ARCHITECTURES

To ease the architecture development [10], the strategy adopted 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, which is specified 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 outside, taking into consideration only the inputs and outputs requirements but differ in the internal details. Based on this observation, the first level of the architecture, called, the external architecture was developed for both 5/3 and 9/7. In the second step, the internal details of the DDGs for each algorithm were considered separately for the development of the processor's datapath architectures, since DDGs internally define and specify the structure of the processors.

In this paper, the first level, the external architectures for 2-parallel and 3-parallel are developed for both 5/3 and 9/7 algorithms based on the scan method shown in Fig.3 [10]. Then the processors' datapath architectures developed in [10] for 5/3 and 9/7 are modified to fit into the two proposed parallel external architectures' processors.

In general, the scan frequency f_l and hence the period $\tau_l = 1/f_l$ 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.

$$\begin{aligned} &\textit{If } t_p / l \cdot k \geq t_m \quad \textit{then} \\ &\qquad \tau_l = t_p / l \cdot k \\ &\textit{else} \quad \tau_l = t_m \end{aligned} \tag{3}$$

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.

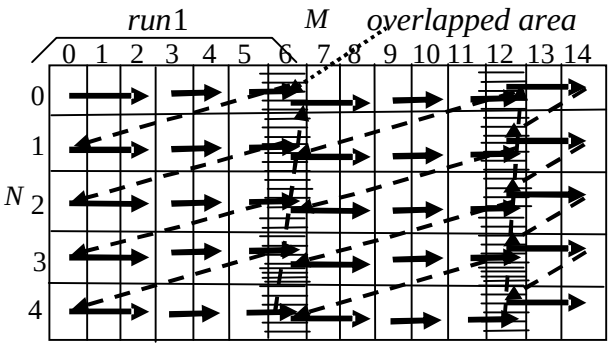
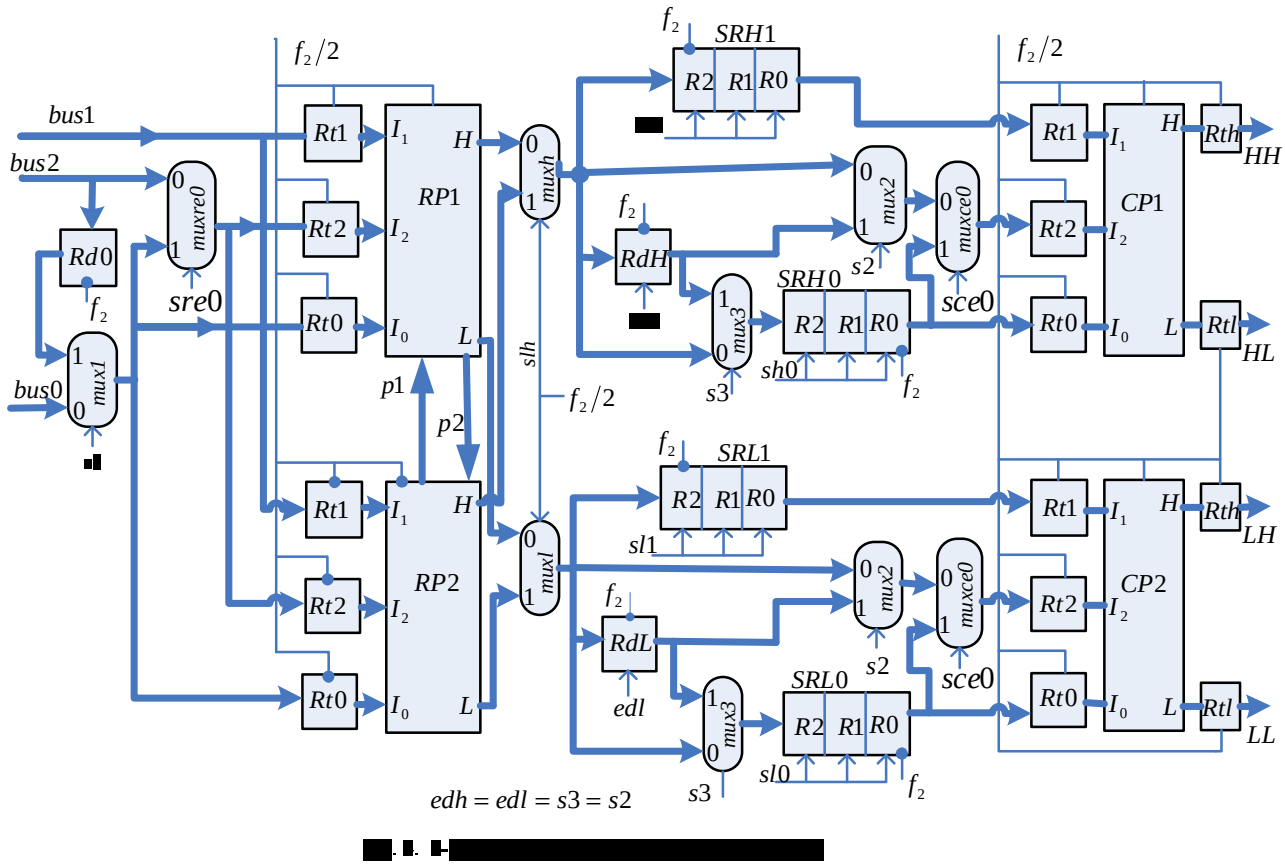


Fig. 3. Third overlapped scan method



A. 2-parallel pipelined intermediate architecture

Based on the DDGs for 5/3 and 9/7 filters shown in Figs. 1 and 2, respectively, and the scan method shown in Fig. 3, the 2-parallel pipelined intermediate architecture shown in Fig. 4 is proposed. The dataflow of the architecture is given in Table 1. 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 [10], the RP and the CP for the 5/3 were pipelined into 4 and 3 stages, respectively, whereas, the RP and the CP for the 9/7 were pipelined into 8 and 6 stages, respectively.

The scan method shown in Fig. 3 not only reduces the power consumption of the external memory but also reduces the memory requirements between RPs and CPs to a few registers and allows the CPs to work in parallel with RPs as earlier as possible.

The proposed external parallel architecture operates with frequency $f_2/2$ and scans the external memory with frequency f_2. The buses labeled *bus0*, *bus1*, and *bus2* are used for transferring every clock cycle pixels from external memory to one of the RPs latches labeled Rt0, Rt1, and Rt2. According to the scan method shown in Fig. 3, in the first clock cycle, the 3 buses are used for scanning the first 3 pixels from the first row of the external memory, whereas in the second and third cycles *bus1* and *bus2* are used for scanning two pixels each cycle. The scan then moves to the second row to repeat the process. The RP1 latches load new data (pixels) every time clock $f_2/2$ makes a positive transition, whereas RP2 latches load new data at the negative transitions. Assume the first half cycle of the clocks f_2 and $f_2/2$ are low. On the other hand, both CP1 and CP2 and their associate latches load new data every time clock $f_2/2$ makes a positive transition.

According to the DDGs, in every clock cycle, 3 pixels are required to initiate an operation. And the third pixel is always needed in the next operation. Therefore, register Rd0 is added in Fig. 4 to hold the third pixel for the next operation. Multiplexer *mux1* passes Rd0 to either Rt0 of RP1 or Rt0 of RP2. Register Rd0 loads a new pixel from *bus2*, every time clock f_2 makes a negative transition.

The control signal *s*1 of the multiplexer labeled *mux1* is set 0 in the first clock cycle of f_2 to pass data in *bus0* and is set 1 in the second and third clock cycles to pass Rd0 contents. The above steps are repeated in cycles 4, 5, and 6 and so on.

The multiplexer labeled *muxre0* is an extension multiplexer and 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. According to the DDGs, the pixels at location *X(r,M-2)*, which will be placed in *bus0*, must be allowed to pass through *muxre0* and then should be loaded into Rt2 as well as Rt0. The two multiplexers labeled *muxce0*, located at the CPs side, are also extension multiplexers and perform the same function as that of *muxre0* when DWT is applied column-wise by the CPs. The registers labeled *SRH1*, *SRH0*, *SRL1*, and *SRL0* are FIFO shift registers each holds at any times 3 coefficients. Registers *SRH1*, *SRH0*, and *RdH* are used for storing high coefficients generated by RP1 and RP2, whereas

TABLE 1 DATAFLOW FOR THE ARCHITECTURE *(K=3)*

Ck	RP	Rd0	RP's input latches Rt0 Rt2 Rt1	RdH	SRH0 R2 R1 R0	SRH1 R2 R1 R0	RdL	SRL0 R2 R1 R0	SRL1 R2 R1 R0
1	1	x 0,2	x 0,0 x 0,2 x 0,1						
2	2	x 0,4	x 0,2 x 0,4 x 0,3						
3	1	x 0,6	x 0,4 x 0,6 x 0,5						
4	2	x 1,2	x 1,0 x 1,2 x 1,1						
5	1	x 1,4	x 1,2 x 1,4 x 1,3						
6	2	x 1,6	x 1,4 x 1,6 x 1,5						
7	1	x 2,2	x 2,0 x 2,2 x 2,1		H0,0 ----- -----			L0,0 ----- -----	
8	2	x 2,4	x 2,2 x 2,4 x 2,3		H0,1 H0,0 -----			L0,1 L0,0 -----	
9	1	x 2,6	x 2,4 x 2,6 x 2,5		H0,2 H0,1 H0,0			L0,2 L0,1 L0,0	
10	2	x 3,2	x 3,0 x 3,2 x 3,1			H1,0 ----- -----			L1,0 ----- -----
11	1	x 3,4	x 3,2 x 3,4 x 3,3			H1,1 H1,0 -----			L1,1 L1,0 -----
12	2	x 3,6	x 3,4 x 3,6 x 3,5		H0,2 H0,1 H0,0	H1,2 H1,1 H1,0		L0,2 L0,1 L0,0	L1,2 L1,1 L1,0
13	1	x 4,2	x 4,0 x 4,2 x 4,1		H2,0 H0,2 H0,1	----- H1,2 H1,1		L2,0 L0,2 L0,1	------ L1,2 L1,1
14	2	x 4,4	x 4,2 x 4,4 x 4,3	H2,1	H2,0 H0,2 H0,1	----- H1,2 H1,1	L2,1	L2,0 L0,2 L0,1	------ L1,2 L1,1
15	1	x 4,6	x 4,4 x 4,6 x 4,5	H2,2	H2,1 H2,0 H 0,2	----- H1,2 H1,1	L2,2	L2,1 L2,0 L0,2	------ L1,2 L1,1
16	2	x 5,2	x 5,0 x 5,2 x 5,1	H2,2	H2,1 H2,0 H 0,2	H3,0 ----- H1,2	L2,2	L2,1 L2,0 L0,2	L3,0 ------ L1,2
17	1	x 5,4	x 5,2 x 5,4 x 5,3	------	H2,2 H2,1 H 2,0	H3,1 H3,0 ------	------	L2,2 L2,1 L2,0	L3,1 L3,0 -----
18	2	x 5,6	x 5,4 x 5,6 x 5,5	------	H2,2 H2,1 H 2,0	H3,2 H3,1 H3,0	------	L2,2 L2,1 L2,0	L3,2 L3,1 L3,0
19	1	x 6,2	x 6,0 x 6,2 x 6,1	------	H4,0 H2,2 H2,1	------ H3,2 H3,1	------	L4,0 L2,2 L2,1	------ L3,2 L3,1
20	2	x 6,4	x 6,2 x 6,4 x 6,3	H4,1	H4,0 H2,2 H2,1	------ H3,2 H3,1	L4,1	L4,0 L2,2 L2,1	------ L3,2 L3,1
21	1	x 6,6	x 6,4 x 6,6 x 6,5	H4,2	H4,1 H4,0 H2,2	------ H3,2 H3,1	L4,2	L4,1 L4,0 L2,2	------ L3,2 L3,1
22	2	x 7,2	x 7,0 x 7,2 x 7,1	H4,2	H4,1 H4,0 H2,2	H5,0 ------ H3,2	L4,2	L4,1 L4,0 L2,2	L5,0 ----- L3,2
23	1	x 7,4	x 7,2 x 7,4 x 7,3	------	H4,2 H4,1 H4,0	H5,1 H5,0 ------	------	L4,2 L4,1 L4,0	L5,1 L5,0 -----
24	2	x 7,6	x 7,4 x 7,6 x 7,5	-----	H4,2 H4,1 H4,0	H5,2 H5,1 H5,0	------	L4,2 L4,1 L4,0	L5,2 L5,1 L5,0
25	1	x 8,2	x 8,0 x 8,2 x 8,1	------	H6,0 H4,2 H4,1	------ H5,2 H5,1	------	L6,0 L4,2 L4,1	----- L5,2 L5,1

ck	RP	CP1 &CP2 input latches Rt0 Rt2 Rt1 Rt0 Rt2 Rt1	CP1 & CP2 output latches Rth Rtl Rth Rtl
13	1	H0,0 H2,0 H1,0 L0,0 L2,0 L1,0	
14	2	----------	
15	1	H0,1 H2,1 H1,1 L0,1 L2,1 L1,1	
16	2	----------	
17	1	H0,2 H2,2 H1,2 L0,2 L2,2 L1,2	
18	2	----------	
19	1	H2,0 H4,0 H3,0 L2,0 L4,0 L3,0	HH0,0 HL0,0 LH0,0 LL0,0
20	2	----------	----------
21	1	H2,1 H4,1 H3,1 L2,1 L4,1 L3,1	HH0,1 HL0,1 LH0,1 LL0,1
22	2	----------	----------
23	1	H2,2 H4,2 H3,2 L2,2 L4,2 L3,2	HH0,2 HL0,2 LH0,2 LL0,2
24	2	----------	----------
25	1	H4,0 H6,0 H5,0 L4,0 L6,0 L5,0	HH1,0 HL1,0 LH1,0 LL1,0

SRL1, *SRL0*, and RdL are used for storing low coefficients. These registers all operate with frequency f_2. The control signals *sl0=sh0* and *sl1=sh1* control the operation of the FIFO registers. When these signals are asserted high, the FIFOs shift in new data, otherwise, no shift take place. High coefficients stored in SRH0 and SRH1 are executed by CP1, while CP2 executes low coefficients stored in SRL0 and SRL1.

The operations of the two multiplexers, labeled *muxh* and *muxl*, can be controlled by one control signal labeled *slh*. This control signal is connected to the clock $f_2/2$. When $f_2/2$ is low, both multiplexers pass coefficients generated by RP1, otherwise, pass coefficients generated by RP2.

In the following, the dataflow of the architecture using Table 1 will be described. In the first cycle, 3 pixels are scanned from external memory locations X(0,0), X(0,1), and X(0,2) and are loaded into RP1 latches by the positive transition of clock $f_2/2$ to initiate the first operation. In addition, pixel of location X(0,2) is also loaded into register Rd0, as indicated in Table 1, since it is needed in the second operation. During clock cycle 2, two pixels are scanned from locations X(0,3) and X(0,4) through bus1 and bus2, respectively, and are loaded , along with the pixel in Rd0, into RP2 latches Rt1, Rt2, and Rt1, respectively, to initiate the second operation. Pixel scanned from location X(0,4) is also loaded into Rd0, since it is required in the third operation.

Cycle 3 scans 2 more pixels, located at X(0,5) and X(0,6), and loads them along with contents of register Rd0 into RP1 latches Rt1, Rt2, and Rt0, respectively. In cycle 4, the scanning moves to the second row and scans 3 pixels from locations X(1,0), X(1,1), and X(1,2) and loads them into RP2 latches. The scanning proceeds according to the scan method in Fig. 3, until the last row is reached, say, to complete the first run. In the second run, the scan returns to the first row to consider for scanning the next 7 columns. This process is repeated until the whole image pixels are scanned.

In cycle 7, the first outputs of the first operation, generated by RP1 and labeled H0,0 and L0,0 in the table, are shifted into SRH0 and SRL0, respectively, by the pulse ending the cycle. In cycle 8, the outputs of the second operation H0,1 and L0,1 generated by RP2 are shifted into SRH0 and SRL0, respectively. Whereas, in cycle 9, the results of the third operation H0,2 and L0,2 generated by RP1 are shifted into SRH0 and SRL0, respectively. In cycle 10, the results of the fourth operation H1,0 and L1,0 produced by RP2 are shifted into registers SRH1 and SRL1, respectively. Similarly, in cycles 11 and 12, two more coefficients are shifted into each SRH1 and SRL1, as shown in the table. The 4 FIFO registers each now contains 3 coefficients.

In cycle 13, for the first time, CP1 and CP2 input latches labeled Rt0, Rt1, and Rt2 simultaneously are loaded with the high coefficients H0,0 , H1,0, and H2,0, and the low coefficients L0,0, L2,0, and L1,0, respectively, as shown in Table 1. Note that coefficient H2,0, generated by RP1 during cycle 13, which passes through the multiplexers labeled *muxh*, *mux2*, and *muxce0*, will be loaded into Rt2 of the CP1 by the pulse ending the cycle. Similarly, L2,0 is loaded into Rt2 of CP2. Moreover, coefficients H2,0 and L2,0 are shifted into SRH0 and SRL0, respectively, as shown in Table 1. During cycle 14 no new coefficients are transferred to CP1 and CP2 latches and the coefficients stored in the 4 FIFO shift registers remain unchanged. However, coefficients H2,1 and L2,1 generated by RP2 during cycle 14 are loaded into registers RdH and RdL, respectively.

Cycle 15 generates coefficients H2,2 and L2,2, which are loaded into registers RdH and RdL, respectively, while their contents are shifted into SRH0 and SRL0 by the pulse ending the cycle. Cycle 15 also loads new coefficients into CP1 and CP2 latches, as shown in Table 1, while contents of the two FIFO registers SRH1 and SRL1 remain unchanged.

In cycle 16, coefficients H3,0 and L3,0 are shifted into SRH1 and SRL1 , respectively, while RdH, SRH0, RdL, and SRL0 maintain their contents and no data are scheduled for CPs. The dataflow then proceeds as shown in table 1.

Observe that the dataflow pattern between cycles 13 and 18 in Table 1, especially in the 4 FIFO registers including RdH and RdL, repeats each 6 clock cycles. A careful investigation of Table 1 from cycles 13 to 18 shows that the control signals of the two multiplexers labeled *mux2* and the two multiplexers labeled *mux3* including the control signals *(edh and edl)* of the registers labeled RdH and RdL can all be combined into one signal, *s2*. Moreover, examination of Table 1 shows that the control signals values for signals s*2*, *sl0=sh0*, and *sl1=sh1* starting from cycles 13 to 18 can be as shown in Table 2. These control signal values repeat every 6 clock cycles.

According to the 5/3 DDGs, each coefficient calculated in the first level (step1) is also required in the calculations of two coefficients in the second level (step 2). That implies a high coefficient calculated by RP1 in stage 1 should be passed to stage 2 of RP2 and vice versa. The 9/7 DDGs also shows similar dependencies among coefficients of two levels or steps. Therefore, the path labeled *P1* and *P2* have been added in Fig. 4 so that the two RPs can pass high coefficients to each other. However, this would require the two RPs datapath architectures for 5/3 and 9/7 to be modified as shown in Figs. 5 and 6, respectively. The 9/7 RPs datapath shown in Fig. 6 seems somewhat very complex. Therefore, its dataflow is provided in Table 3, which describes in detail how it works. In this table, the first index refers to a row number, while the second refers to a pixel or a coefficient number in the DDGs. Note that in Table 3 at cycle 5 coefficient $\tilde{Y}0,5$ is stored in the first location in TLB1 of RP1, whereas at cycle 11 coefficient $\tilde{Y}2,5$ is stored in the second location.

In addition, if the third high coefficient of the first row labeled Y(5) in the 5/3 DDGs is stored in the first location in TLB1 of RP1, then the third high coefficient of the second row should be stored in the first location in TLB1 of RP2 and so on. Similarly, the 9/7 coefficients labeled $Y''(5)$, $Y''(4)$, and $Y'(3)$ in the DDGs generated by processing the first row of the first run should be stored in the first locations of each TLB1, TLB2, and TLB3 of RP1, respectively. On the other hand, the 3 coefficients generated by processing the second row of the first run should be stored in the first locations of each TLB1, TLB2, and TLB3 of RP2, respectively, and so on. The same process also applies in all subsequent runs.

Based on Table 3, the control signal *sf* of the 8 multiplexers labeled *muxf* in Fig. 6 can be set 0 in the first run and 1 in all other runs. It is very important to note that, especially in the first run, the scan method in Fig. 3 allows 5/3 RPs to yield 6 coefficients, half belong to the first 3 columns of H decomposition and the other half to L decomposition, each time it processes 7 pixels of each row. On the other hand, the 9/7 yields only 4 coefficients, 2 high and 2 low, by processing the same number of pixels in each row. This implies that in the first run each 5/3 CP would process 3 columns in an interleave fashion as shown in Table 1, whereas each 9/7 CP would process in the first run only two columns in an interleave fashion. However, in all subsequent runs, except the last, each 9/7 CP and each 5/3 CP would process 3 columns at a time. This interleaving process, however, would require 9/7 and 5/3 CPs to be modified in order to allow interleaving in execution to take place.

The advantage of this organization is that the TLBs in Figs. 5 and 6 are not required to be read and written in the same

TABLE 2
CONTROL SIGNAL VALUES FOR *s3*, *sl0*, AND *sl1*

Cycle number	*s2*	*sl0=sh0*	*sl1=sh1*
13	0	1	1
14	1	0	0
15	1	1	0
16	0	0	1
17	1	1	1
18	0	0	1

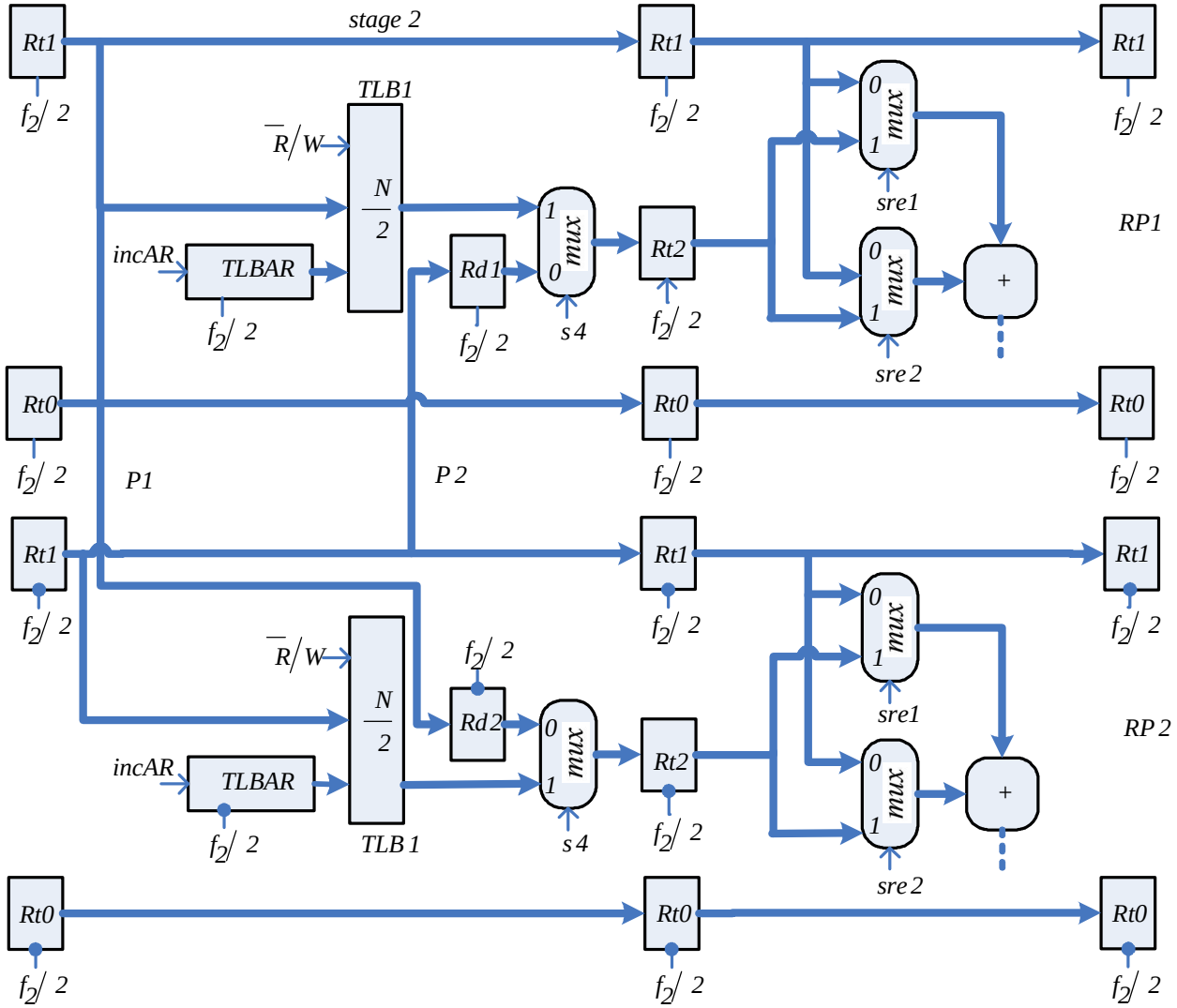


Fig. 5. Modified stage 2 of 5/3 RPS datapath architecture

clock cycle. Because, according to the scan method shown in Fig. 3, 7 pixels are scanned from each row to initiate 3 successive operations and the TLB is read in the first operation and is written in the third operation starting from the second run. Furthermore, the fact that 7 pixels are scanned from each row to initiate 3 consecutive operations and the TLB is read in the first operation and written in the third can be used to derive, for all runs except the last, the control signal values for the signals labeled $\overline{R}/W$ and *incAR* in both 5/3 and 9/7 TLBs including *s4*, as shown in Table 4. These signal values repeat every 3 cycles starting from the first cycle. Signals in Table 4, including the extension multiplexers control signals, can be carried by latches, similar to pipeline latches, from the control unit to the first stage of the pipeline then to the next stage and so on. When a stage where a signal(s) is used is reached that signal(s) can be dropped and the rest are carried on to the next stage and so on until they are all used.

B. Transitions to the last run

The description given so far including the control signal values in Tables 2 and 4 apply to all runs except the last run, which requires special handling. The last run in any decomposition level can be determined and detected by subtracting after each run 6 from the width *(M)* of an image. The last run is reached when *M* becomes less than or equal to 6 *(M≤6)* and *M* can have one of the six different values 6, 5, 4, 3, 2, or 1, which imply 6 different cases. These values give number of external memory columns that will be considered for scanning in the last run.

According to the scan method, in each run, 7 columns of the external memory are considered for scanning and each 7 pixels scanned, one from each column, initiate 3 consecutive operations. Thus, since, cases 6 and 5 initiate 3 operations; they can be handled as normal runs.

On the other hand, cases 4 and 3 initiate 2 operations and the dataflow in the last run will differ from the normal dataflow given in Table 1. Therefore, 2 dataflow are provided in Tables 5 and 6 for even and odd *N*, respectively, so that they can be applied when either of the two cases occurs. The dataflow shown in Table 5 is derived for case 4 but it can be used also for case3. Similarly, Table 6 is derived for case3 but it can be used also for case 4. Moreover, examination of Tables 5 and 6 show that after *2k+2* cycles from the last

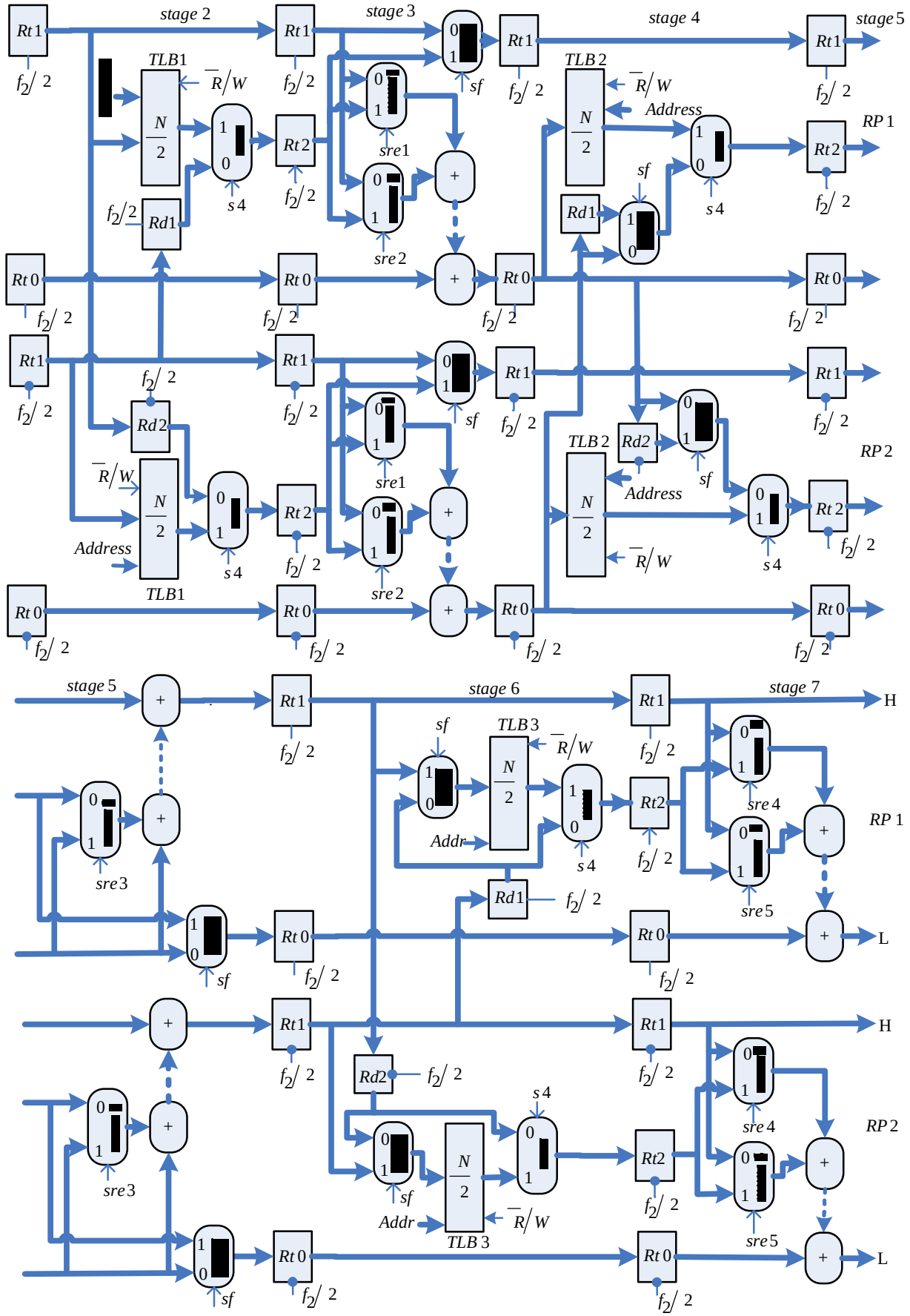


Fig. 6. Modified 9/7 RPs datapath for 2-parallel intermediate architecture

TABLE 3 DATAFLOW FOR MODIFIED 9/7 RPS DATAPATH (FIG. 6)

RP1						
	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7
CK	Rt0 Rt1 Rd1 TLB1	Rt0 Rt2 Rt1	Rt0 Rt1 Rd1 TLB2	Rt0 Rt2 Rt1	Rt0 Rt1 Rd1 TLB3	Rt0 Rt2 Rt1
1	X0,0 Ỹ0,1 --- -----					
3	X0,4 Ỹ0,5 Ỹ0,3 ----	X0,0 ---- Ỹ0,1				
5	X1,2 Ỹ1,3 Ỹ1,1 Ỹ0,5	X0,4 Ỹ0,3 Ỹ0,5	Ỹ0,0 Ỹ0,1 ---- ----			
7	X2,0 Ỹ2,1 Ỹ1,5	X1,2 Ỹ1,1 Ỹ1,3	Ỹ0,4 ---- ----- ----	Ỹ0,0 Ỹ0,2 Ỹ0,1		
9	X2,4 Ỹ2,5 Ỹ2,3	X2,0 ---- Ỹ2,1	Ỹ1,2 Ỹ1,3 ---- Ỹ0,4	---- ---- ----	Ỹ0,0 Ý0,1 ---- -----	
11	---------- Ỹ2,5	X2,4 Ỹ2,3 Ỹ2,5	Ỹ2,0 Ỹ2,1 ----	Ỹ1,2 Ỹ1,4 Ỹ1,3	----- ----- Ý0,3 ----	Ỹ0,0 ---- Ý0,1
13	X0,6 Ỹ0,7 -----	----------	Ỹ2,4 ---- ----	Ỹ2,0 Ỹ2,2 Ỹ2,1	Ỹ1,2 Ý1,3 Ý1,1 Ý0,3	----- ---- -----
15	X0,10 Ỹ0,11Ỹ0,9	X0,6 Ỹ0,5 Ỹ0,7	---------- Ỹ2,4	----------	Ỹ2,0 Ý2,1 ---- -----	Ỹ1,2 Ý1,1 Ý1,3
17	X1,8 Ỹ1,9 Ỹ1,7 Ỹ0,11	X0,10 Ỹ0,9 Ỹ0,11	Ỹ0,6 Ỹ0,5 ----	----------	---------- Ý2,3	Ỹ2,0 ----- Ý2,1
19	X2,6 Ỹ2,7 ------	X1,8 Ỹ1,7 Ỹ1,9	Ỹ0,10 Ỹ0,9 Ỹ0,8	Ỹ0,6 Ỹ0,4 Ỹ0,5	---------- Ý2,3	----------
21	X2,10 Ỹ2,11 Ỹ2,9	X2,6 Ỹ2,5 Ỹ2,7	Ỹ1,8 Ỹ1,7 Ỹ1,6 Ỹ0,10	Ỹ0,10 Ỹ0,8 Ỹ0,9	Ỹ0,4 Ý0,5 ----	----------
23	Ỹ2,11	X2,10 Ỹ2,9 Ỹ2,11	Ỹ2,6 Ỹ2,5 ----	Ỹ1,8 Ỹ1,6 Ỹ1,7	Ỹ0,8 Ý0,9 Ý0,7	Ỹ0,4 Ý0,3 Ý0,5
25			Ỹ2,10 Ỹ2,9 Ỹ2,8	Ỹ2,6 Ỹ2,4 Ỹ2,5	Ỹ1,6 Ý1,7 Ý1,5 Ý0,9	Ỹ0,8 Ý0,7 Ý0,9
27			Ỹ2,10	Ỹ2,10 Ỹ2,8 Ỹ2,9	Ỹ2,4 Ý2,5 ----	Ỹ1,6 Ý1,5 Ý1,7
29					Ỹ2,8 Ý2,9 Ý2,7	Ỹ2,4 Ý2,3 Ý2,5
31					Ý2,9	Ỹ2,8 Ý2,7 Ý2,9
33						----------
RP2						
	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7
CK	Rt0 Rt1 Rd2 TLB1	Rt0 Rt2 Rt1	Rt0 Rt1 Rd2 TLB2	Rt0 Rt2 Rt1	Rt0 Rt1 Rd2 TLB3	Rt0 Rt2 Rt1
2	X0,2 Ỹ0,3 Ỹ0,1 ----					
4	X1,0 Ỹ1,1 Ỹ0,5 ----	X0,2 Ỹ0,1 Ỹ0,3				
6	X1,4 Ỹ1,5 Ỹ1,3 ----	X1,0 ---- Ỹ1,1	Ỹ0,2 Ỹ0,3 ---- ----			
8	X2,2 Ỹ2,3 Ỹ2,1 Ỹ1,5	X1,4 Ỹ1,3 Ỹ1,5	Ỹ1,0 Ỹ1,1 ----- ----	Ỹ0,2 Ỹ0,4 Ỹ0,3		
10	----------	X2,2 Ỹ2,1 Ỹ2,3	Ỹ1,4 ---- ---- ----	Ỹ1,0 Ỹ1,2 Ỹ1,1	Ỹ0,2 Ý0,3 Ý0,1 ----	
12	----------	----------	Ỹ2,2 Ỹ2,3 ---- Ỹ1,4	----- ----- ----	Ỹ1,0 Ý1,1 ---- ----	Ỹ0,2 Ý0,1 Ý0,3
14	X0,8 Ỹ0,9 Ỹ0,7	----------	----------	Ỹ2,2 Ỹ2,4 Ỹ2,3	---- ---- Ý1,3 ----	Ỹ1,0 ----- Ý1,1
16	X1,6 Ỹ1,7 -----	X0,8 Ỹ0,7 Ỹ0,9	----------	----------	Ỹ2,2 Ỹ2,3 Ý2,1 Ý1,3	----------
18	X1,10 Ỹ1,11Ỹ1,9	X1,6 Ỹ1,5 Ỹ1,7	Ỹ0,8 Ỹ0,7 Ỹ0,6	----------	----------	Ỹ2,2 Ý2,1 Ý2,3
20	X2,8 Ỹ2,9 Ỹ2,7 Ỹ1,11	X1,10 Ỹ1,9 Ỹ1,11	Ỹ1,6 Ỹ1,5 ----	Ỹ0,8 Ỹ0,6 Ỹ0,7	----------	----------
22		X2,8 Ỹ2,7 Ỹ2,9	Ỹ1,10 Ỹ1,9 Ỹ1,8	Ỹ1,6 Ỹ1,4 Ỹ1,5	Ỹ0,6 Ý0,7 Ý0,5	----------
24			Ỹ2,8 Ỹ2,7 Ỹ2,6 Ỹ1,10	Ỹ1,10 Ỹ1,8 Ỹ1,9	Ỹ1,4 Ý1,5 -----	Ỹ0,6 Ý0,5 Ý0,7
26				Ỹ2,8 Ỹ2,6 Ỹ2,7	Ỹ1,8 Ý1,9 Ý1,7	Ỹ1,4 Ý1,3 Ý1,5
28					Ỹ2,6 Ý2,7 Ý2,5 Ý1,9	Ỹ1,8 Ý1,7 Ý1,9
30						Ỹ2,6 Ý2,5 Ý2,7
32						----------

TABLE 4 CONTROL SIGNAL VALUES FOR SIGNALS IN STAGE 2 OF BOTH RP1 AND RP2

Cycle Number	RP number	$\overline{R}/W$	*incAR*	*s4*
1	1	0	0	1
2	2	0	0	0
3	1	1	1	0

empty cycle, where *k* is the number of pipeline stages of the RPs, the control signal values for signals *s2, sl0*, and *sl1*, which repeat every 4 clock cycles, should be as shown in Table 7 for the rest of the decomposition level. However, during the *2k+2* and the empty cycles, the control signal values for *s2, sl0* and *sl1* follow Table 3. Therefore, cases 4 and 3 can be considered as one case. Only at the beginning of the transition to the last run, if *N* is even, then one empty cycle is inserted, otherwise,
4 cycles are inserted, according to Table 5 and 6, respectively. During an Empty cycle external memory is not scanned as in cycle 25 in Table 5.

On the other hand, cases 2 and 1, each initiate one operation. Case 2 initiates an operation each time 2 pixels, one from each column, are scanned, whereas case 1 initiate an operation each time a pixel is scanned from the last column. Therefore, dataflow of the last run in the two cases will differ from the normal dataflow given in Table 1. Similar to cases 4 and 3, two dataflow for cases 2 and 1 can be derived and the control signal values for signals *s2, sl0*, and *sl*1 can be determined by investigating the two dataflow. The result of the investigation would show that the control signal values for both cases 2 and 1 are identical; therefore, the two cases can be treated as one case.

Fig. 7 shows the block diagram of the control unit that generates signals *s2, sl0*, and *sl1* along with the circuits that detect the occurrence of the last run and the 6 cases. First, *M* is loaded into register RM, then register R6, which contain the 2's complement of 6, is subtracted from RM through the 2's complement adder circuit and the result of the subtraction is loaded back into RM. If *Lr* is 1, then that implies the last run is reached and the result of the subtraction is not transferred to RM. The 3 least significant bits of register RM is then examined by the control unit to determine which of the 6 cases has occurred. First *z1* is examined. If *z1* is 1, that implies the occurrence of either cases 6 or 5 and the control unit proceeds as usual. But, if *z*1 is 0, then *z2* is examined. If z2 is 1, then cases 4 and 3 are applied, otherwise, cases 2 and 1 are applied.

The above description can be generalized for determining

TABLE 5 DATAFLOW OF THE LAST RUN FOR CASES 4 AND 3 WHEN N IS EVEN

Ck	RP	Rd0	RP's input latches Rt0 Rt2 Rt1	RdH	SRH0 R2 R1 R0	SRH1 R2 R1 R0	RdL	SRL0 R2 R1 R0	SRL1 R2 R1 R0
25	1	------	----------	------	H6,0 H4,2 H4,1	------ H5,2 H5,1	-----	L6,0 L4,2 L4,1	----- L5,2 L5,1
26	2	x 0,8	x 0,6 x 0,8 x 0,7	H6,1	H6,0 H4,2 H4,1	------ H5,2 H5,1	L6,1	L6,0 L4,2 L4,1	----- L5,2 L5,1
27	1	------	x 0,8 x 0,8 x 0,9	H6,2	H6,1 H6,0 H4,2	------ H5,2 H5,1	L6,2	L6,1 L6,0 L4,2	----- L5,2 L5,1
28	2	x 1,8	x 1,6 x 1,8 x 1,7	H6,2	H6,1 H6,0 H4,2	H7,0 ------ H5,2	L6,2	L6,1 L6,0 L4,2	L7,0 ------ L5,2
29	1	------	x 1,8 x 1,8 x 1,9	-----	H6,2 H6,1 H6,0	H7,1 H7,0 ------	------	L6,2 L6,1 L6,0	L7,1 L7,0 ------
30	2	x 2,8	X2,6 x 2,8 x 2,7	------	H6,2 H6,1 H6,0	H7,2 H7,1 H7,0	------	L6,2 L6,1 L6,0	L7,2 L7,1 L7,0
31	1	------	x 2,8 x 2,8 x 2,9	------	----- H6,2 H6,1	------ H7,2 H7,1	------	------ L6,2 L6,1	------ L7,2 L7,1
32	2	x 3,8	x 3,6 x 3,8 x 3,7	H0,3	----- H6,2 H6,1	------ H7,2 H7,1	L0,3	------ L6,2 L6,1	------ L7,2 L7,1
33	1	------	x 3,8 x 3,8 x 3,9	H0,4	H0,3 ----- H6,2	------ H7,2 H7,1	L0,4	L0,3 ------ L6,2	------ L7,2 L7,1
34	2	x 4,8	x 4,6 x 4,8 x 4,7	H0,4	H0,3 ----- H6,2	H1,3 ------ H7,2	L0,4	L0,3 ----- L6,2	L1,3 ------ L7,2
35	1	------	x 4,8 x 4,8 x 4,9	------	H0,4 H0,3 -----	H1,4 H1,3 ------	------	L0,4 L0,3 -----	L1,4 L1,3 -----
36	2	x 5,8	x 5,6 x 5,8 x 5,7	H2,3	----- H0,4 H0,3	------ H1,4 H1,3	L2,3	----- L0,4 L0,3	------ L1,4 L1,3
37	1	------	x 5,8 x 5,8 x 5,9	H2,4	H2,3 ----- H0,4	------ H1,4 H1,3	L2,4	L2,3 ----- L0,4	------ L1,4 L1,3
38	2	x 6,8	x 6,6 x 6,8 x 6,7	H2,4	H2,3 ----- H0,4	H3,3 ------ H1,4	L2,4	L2,3 ----- L0,4	L3,3 ------ L1,4
39	1	------	x 6,8 x 6,8 x 6,9	------	H2,4 H2,3 -----	H3,4 H3,3 ------	------	L2,4 L2,3 -----	L3,4 L3,3 -----
40	2	x 7,8	x 7,6 x 7,8 x 7,7	H4,3	------ H2,4 H2,3	----- H3,4 H3,3	L4,3	------ L2,4 L2,3	----- L3,4 L3,3
41	1	------	x 7,8 x 7,8 x 7,9	H4,4	H4,3 ------ H2,4	----- H3,4 H3,3	L4,4	L4,3 ------ L2,4	----- L3,4 L3,3
42	2	------	----------	H4,4	H4,3 ------ H2,4	H5,3 ----- H3,4	L4,4	L4,3 ------ L2,4	L5,3 ----- L3,4
43	1	------	----------	------	H4,4 H4,3 ------	H5,4 H5,3 -----	------	L4,4 L4,3 ------	L5,4 L5,3 -----
44	2	------	----------	H6,3	----- H4,4 H4,3	------ H5,4 H5,3	L6,3	----- L4,4 L4,3	------ L5,4 L5,3
45	1	------	----------	H6,4	H6,3 ----- H4,4	------ H5,4 H5,3	L6,4	L6,3 ----- L4,4	------ L5,4 L5,3
46	2	------	----------	H6,4	H6,3 ----- H4,4	H7,3 ------ H5,4	L6,4	L6,3 ----- L4,4	L7,3 ------ L5,4
47	1	------	----------	------	H6,4 H6,3 -----	H7,4 H7,3 ------	------	L6,4 L6,3 -----	L7,4 L7,3 -----
48	2	------	----------	------	----- H6,4 H6,3	----- H7,4 H7,3	------	----- L6,4 L6,3	----- L7,4 L7,3
49	1	------	----------	------	----- ------ H6,4	----- H7,4 H7,3	------	----- ------ L6,4	----- L7,4 L7,3
50	2	------	----------	------	----- ------ H6,4	----- ------ H7,4	------	----- ------ L6,4	----- ------ L7,4

ck	RP	CP1 &CP2 input latches Rt0 Rt2 Rt1 Rt0 Rt2 Rt1	CP1 & CP2 output latches Rt Rt Rt Rt
29	1	H4,2 H6,2 H5,2 L4,2 L6,2 L5,2	HH1,2 HL1,2 LH1,2 LL1,2
30	2	----------	----------
31	1	H6,0 H6,0 H7,0 L6,0 L6,0 L7,0	HH2,0 HL2,0 LH2,0 LL2,0
32	2	----------	----------
33	1	H6,1 H6,1 H7,1 L6,1 L6,1 L7,1	HH2,1 HL2,1 LH2,1 LL2,1
34	2	----------	----------
35	1	H6,2 H6,2 H7,2 L6,2 L6,2 L7,2	HH2,2 HL2,2 LH2,2 LL2,2
36	2	----------	----------
37	1	H0,3 H2,3 H1,3 L0,3 L2,3 L1,3	HH3,0 HL3,0 LH3,0 LL3,0
38	2	----------	----------
39	1	H0,4 H2,4 H1,4 L0,4 L2,4 L1,4	HH3,1 HL3,1 LH3,1 LL3,1
40	2	----------	----------
41	1	H2,3 H4,3 H3,3 L2,3 L4,3 L3,3	HH3,2 HL3,2 LH3,2 LL3,2
42	2	----------	----------
43	1	H2,4 H4,4 H3,4 L2,4 L4,4 L3,4	HH0,3 HL0,3 LH0,3 LL0,3
44	2	----------	----------
45	1	H4,3 H6,3 H5,3 L4,3 L6,3 L5,3	HH0,4 HL0,4 LH0,4 LL0,4
46	2	----------	----------
47	1	H4,4 H6,4 H5,4 L4,4 L6,4 L5,4	HH1,3 HL1,3 LH1,3 LL1,3
48	2	----------	----------
49	1	H6,3 H6,3 H7,3 L6,3 L6,3 L7,3	HH1,4 HL1,4 LH1,4 LL1,4
50	2	----------	----------
51	1	H6,4 H6,4 H7,4 L6,4 L6,4 L7,4	HH2,3 HL2,3 LH2,3 LL2,3

TABLE 6 DATAFLOW OF THE LAST RUN FOR CASES 4 AND 3 WHEN N IS ODD

Ck	RP	Rd0	RP's input latches Rt0 Rt2 Rt1	RdH	SRH0 R2 R1 R0	SRH1 R2 R1 R0	RdL	SRL0 R2 R1 R0	SRL1 R2 R1 R0
22	2	------	----------	H4,2	H4,1 H4,0 H2,2	H5,0 ------ H3,2	L4,2	L4,1 L4,0 L2,2	L5,0 ----- L3,2
23	1	------	----------	------	H4,2 H4,1 H4,0	H5,1 H5,0 ------	------	L4,2 L4,1 L4,0	L5,1 L5,0 ----
24	2	------	----------	------	H4,2 H4,1 H4,0	H5,2 H5,1 H5,0	------	L4,2 L4,1 L4,0	L5,2 L5,1 L5,0
25	1	------	----------	------	H6,0 H4,2 H4,1	----- H5,2 H5,1	------	L6,0 L4,2 L4,1	----- L5,2 L5,1
26	2	x 0,8	x 0,6 x 0,8 x 0,7	H6,1	H6,0 H4,2 H4,1	----- H5,2 H5,1	L6,1	L6,0 L4,2 L4,1	----- L5,2 L5,1
27	1	------	x 0,8 ------ -------	H6,2	H6,1 H6,0 H4,2	----- H5,2 H5,1	L6,2	L6,1 L6,0 L4,2	----- L5,2 L5,1
28	2	x 1,8	x 1,6 x 1,8 x 1,7	H6,2	H6,1 H6,0 H4,2	----- ------ H5,2	L6,2	L6,1 L6,0 L4,2	----- ----- L5,2
29	1	------	x 1,8 ------ -------	------	H6,2 H6,1 H6,0	----------	------	L6,2 L6,1 L6,0	----------
30	2	x 2,8	X2,6 x 2,8 x 2,7	------	H6,2 H6,1 H6,0	----------	------	L6,2 L6,1 L6,0	----------
31	1	------	x 2,8 ------ ------	------	------ H6,2 H6,1	----------	------	------ L6,2 L6,1	----------
32	2	x 3,8	x 3,6 x 3,8 x 3,7	H0,3	------ H6,2 H6,1	----------	L0,3	------ L6,2 L6,1	----------
33	1	------	x 3,8 ------ -------	------	H0,3 ------ H6,2	----------	L0,4	L0,3 ------ L6,2	----------
34	2	x 4,8	x 4,6 x 4,8 x 4,7	------	H0,3 ------ H6,2	H1,3 ------ ----	L0,4	L0,3 ------ L6,2	L1,3 ----- -----
35	1	------	x 4,8 ------ -------	------	------ H0,3 ------	------ H1,3 ----	------	L0,4 L0,3 ------	L1,4 L1,3 -----
36	2	x 5,8	x 5,6 x 5,8 x 5,7	H2,3	------ ------ H0,3	------ ----- H1,3	L2,3	------ L0,4 L0,3	------ L1,4 L1,3
37	1	------	x 5,8 ------ ------	------	H2,3 ------ -----	------ ----- H1,3	L2,4	L2,3 ------ L0,4	------ L1,4 L1,3
38	2	x 6,8	x 6,6 x 6,8 x 6,7	------	H2,3 ------ -----	H3,3 ------ -----	L2,4	L2,3 ------ L0,4	L3,3 ----- L1,4
39	1	------	x 6,8 ------ ------	------	------ H2,3 -----	------ H3,3 -----	------	L2,4 L2,3 ------	L3,4 L3,3 -----
40	2	------	----------	H4,3	------ ------ H2,3	------ ------ H3,3	L4,3	------ L2,4 L2,3	----- L3,4 L3,3
41	1	------	----------	------	H4,3 ------- -----	------ ------ H3,3	L4,4	L4,3 ------ L2,4	----- L3,4 L3,3
42	2	------	----------	------	H4,3 ------- -----	H5,3 ------ -----	L4,4	L4,3 ------ L2,4	L5,3 ---- L3,4

ck	RP	CP1 &CP2 input latches Rt0 Rt2 Rt1 Rt0 Rt2 Rt1	CP1 & CP2 output latches Rt Rt Rt Rt
22	2	----------	----------
23	1	H2,2 H4,2 H3,2 L2,2 L4,2 L3,2	HH0,2 HL0,2 LH0,2 LL0,2
24	2	----------	----------
25	1	H4,0 H6,0 H5,0 L4,0 L6,0 L5,0	HH1,0 HL1,0 LH1,0 LL1,0
26	2	----------	----------
27	1	H4,1 H6,1 H5,1 L4,1 L6,1 L5,1	HH1,1 HL1,1 LH1,1 LL1,1
28	2	----------	----------
29	1	H4,2 H6,2 H5,2 L4,2 L6,2 L5,2	HH1,2 HL1,2 LH1,2 LL1,2
30	2	----------	----------
31	1	H6,0 ------ ------ L6,0 ------ ------	HH2,0 HL2,0 LH2,0 LL2,0
32	2	----------	----------
33	1	H6,1 ------ ------ L6,1 ----- ------	HH2,1 HL2,1 LH2,1 LL2,1
34	2	----------	----------
35	1	H6,2 ------ ------ L6,2 ----- ------	HH2,2 HL2,2 LH2,2 LL2,2
36	2	----------	----------
37	1	H0,3 H2,3 H1,3 L0,3 L2,3 L1,3	------- HL3,0 ------- LL3,0
38	2	----------	----------
39	1	---------- L0,4 L2,4 L1,4	------- HL3,1 ------- LL3,1
40	2	----------	----------
41	1	H2,3 H6,4 H3,3 L2,3 L4,3 L3,3	-------- HL3,2 -------- LL3,2
42	2	----------	----------

the last run for any scan method (first, second, or third scan method and so on) used in designing single or *l*-parallel architectures. Thus, in general, the last run in any scan method can be determined and detected by subtracting after each run *2i* from the width *(M)* of an image. The last run is reached when *M* becomes less than or equal *2i (M≤2i)*, where *i=1, 2, 3…* denote the first, the second, and the third scan methods. *M* can have one of *2i* different values, when last run is reached, as follows: *2i, 2i*-1, *2i-2 … 2,* 1, which implies *2i* cases. These values give number of external memory columns that would be considered for scanning in the last run. In addition, cases *2i* and *2i*-1 can always be handled as normal runs.

C. 3-parallel pipelined intermediate architecture

The 2-parallel pipelined intermediate architecture can be extended to 3-parallel pipelined intermediate architecture as shown in Fig. 8. This architecture increases the speed up by

TABLE 7 CONTROL SIGNAL VALUES FOR s2, sl0, AND sl1, IN THE LAST RUN.

Cycle number	s2	*sl0=sh0*	*sl1=sh1*
34	0	0	1
35	1	1	1
36	1	1	1
37	1	1	0

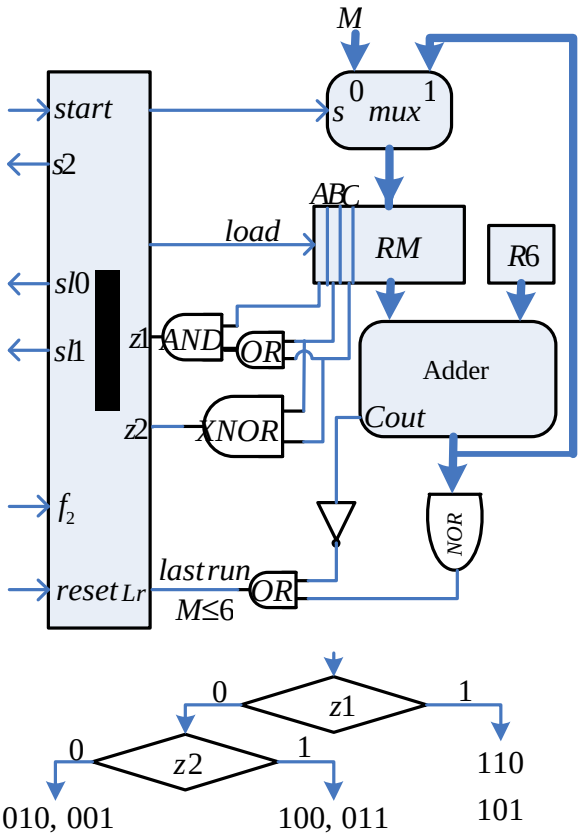


Fig. 7. A control circuit that determines the last run

factor of 3 as compared with single pipelined architecture. The architecture performs its computations according to the dataflow given in Table 8. It operates with frequency $f_3/3$ and scans the external memory with frequency f_3. The clock frequency f_3 can be obtained from (3) as

$$f_3 = 3k/t_p \quad (4)$$

The waveform of the frequency f_3 including the two waveforms of the frequency $f_3/3$ labeled f_{3a} and f_{3b} that can be generated from f_3 are shown in Fig. 9.

The RP2 loads new data into its latches every time clock f_{3b} makes a positive transition, whereas RP1 and RP3 load when clock f_{3a} makes a positive and a negative transition, respectively. On the other hand, CP1 and CP3 loads simultaneously new data every time clock f_{3a} makes a positive transition and CP2 loads every time clock f_{3b} makes a positive transition. Furthermore, for the architecture to operate properly, the three clocks labeled f_3, f_{3a}, and f_{3b} must be synchronized as shown in Fig.9. Clock f_{3a} and f_{3b} can be generated from f_3 using a 2-bit register clocked by f_3 and with a synchronous control signal *clear*. To obtain the divide-by-3 frequency, the register should be designed to count from 0 to 2 and then repeats. The synchronization can then be achieved by the control unit simply by asserting the *clear* signal high just before the first cycle where the external memory scanning begins.

The buses labeled *bus0*, *bus1*, and *bus2* are used for transferring, in every clock cycle, 3 pixels from external memory to one of the RPs latches labeled Rt0, Rt1, and Rt2. In the first clock cycle, 3 pixels are scanned from external memory, locations *X(0,0)*, *X(0,1)* and *X(0,2)*, and are loaded into RP1 latches to initiate the first operation. While the third pixel *(X(0,2))* in *bus2*, which is required in the next operation, is also loaded into Rd0. The second clock cycle scans 2 pixels from external memory, locations *X(0,3)* and *X(0,4)*, through *bus1* and *bus2* , respectively, and loads them into RP2 latches along with the pixel in register Rd0 by the positive transition of clock f_{3a}. This cycle also stores pixel carried by *bus2* in register Rd0. Similarly, the third clock cycle transfers 2 pixels from external memory, locations *X(0,5)* and *X(0,6)*, including the pixel in register Rd0 to RP3 latches to initiate the third operation. The scan then moves to the second row to repeat the process starting from RP1, as shown in Table 8.

The paths labeled *P1*, *P2*, and *P3* in Fig. 8 are used for passing coefficients between the three RPs, since a coefficient calculated in one stage of a RP is always required in the next stage of another RP. This will require the combined three RPs datapath architectures for 5/3 and 9/7 to be modified as shown in Figs. 10 (a) and (a, b), respectively, so that they can fit into RPs of the 3-parallel architecture shown in Fig. 8. Note that Figs. 10 (a) and (b) together form the 9/7 RPs datapath architecture. This architecture can be verified using the 9/7 DDGs shown in Fig. 2. The control signal *sf* of the 9 multiplexers, labeled *muxf* in Fig. 10, is set 1 in the first run and 0 in all other runs.

In the 5/3 datapath architecture shown in Fig. 10 (a), all high coefficients calculated in stage 1 of the RP3 in each run are stored in TLB of stage 2 so that they can be used by RP1 in the calculations of low coefficients in the next run. On the other hand, the 9/7 datapath stores the coefficients labeled Y"(5), Y"(4), and Y'(3) in the DDGs, which can be generated by processing every row of the first run, in TLB1, TLB2, and TLB3, respectively. Similarly, all other runs can be handled.

Similar to the 2-parallel architecture, the 5/3 RPs will generate 6 coefficients each time 7 pixels of each row are processed. While, the 9/7 RPs will generate 4 coefficients by processing the same number of pixels in the first run. Each 4 coefficients will be generated by RP1 and RP2, while RP3 will generate invalid coefficients during the first run. As shown in Table 8, each CP in the 3-parallel architecture processes, in each run, 2 columns coefficients in an interleave fashion. This interleave processing will also require each CP to be modified to allow interleaving in execution.

In the first run the TLB is only written. However, starting from the second run until the run before last, the TLB is read and written in the same clock cycle, with respect to clock f_{3a}. The negative transition of clock f_{3a} always brought a new high coefficient from stage 1 into stage 2 of the RP3. During the low pulse of clock f_{3a} the TLB is read and the result is placed

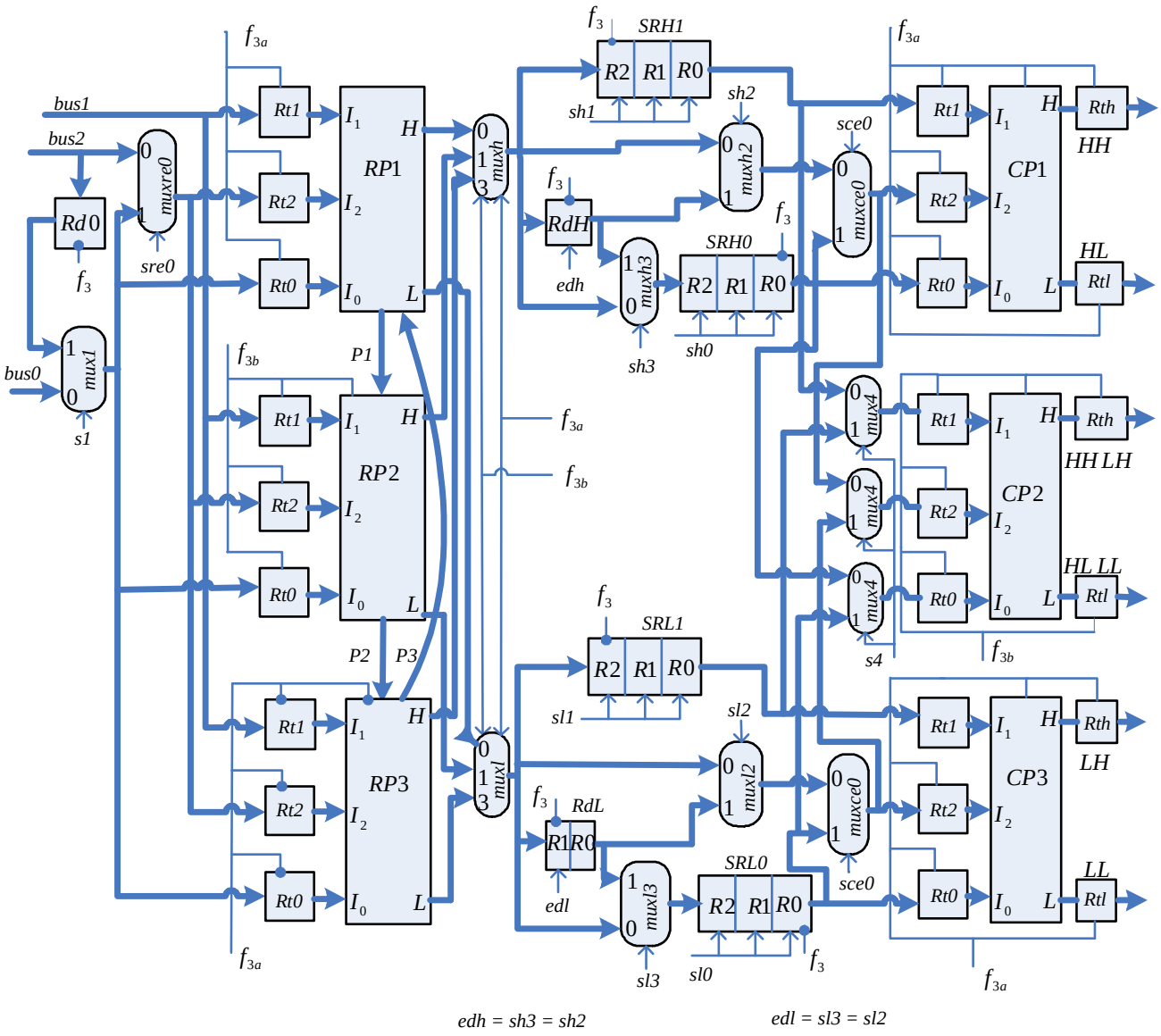


Fig. 8. 3-parallel pipelined intermediate architecture

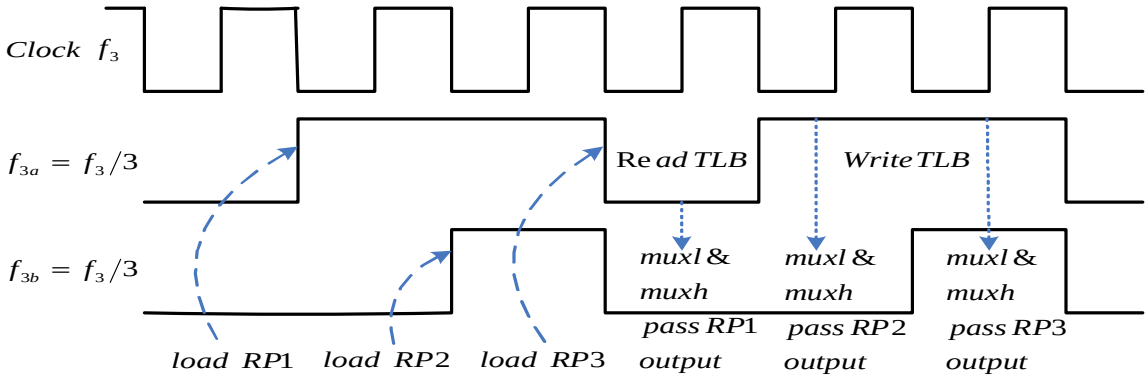


Fig. 9. Waveforms of the three clocks

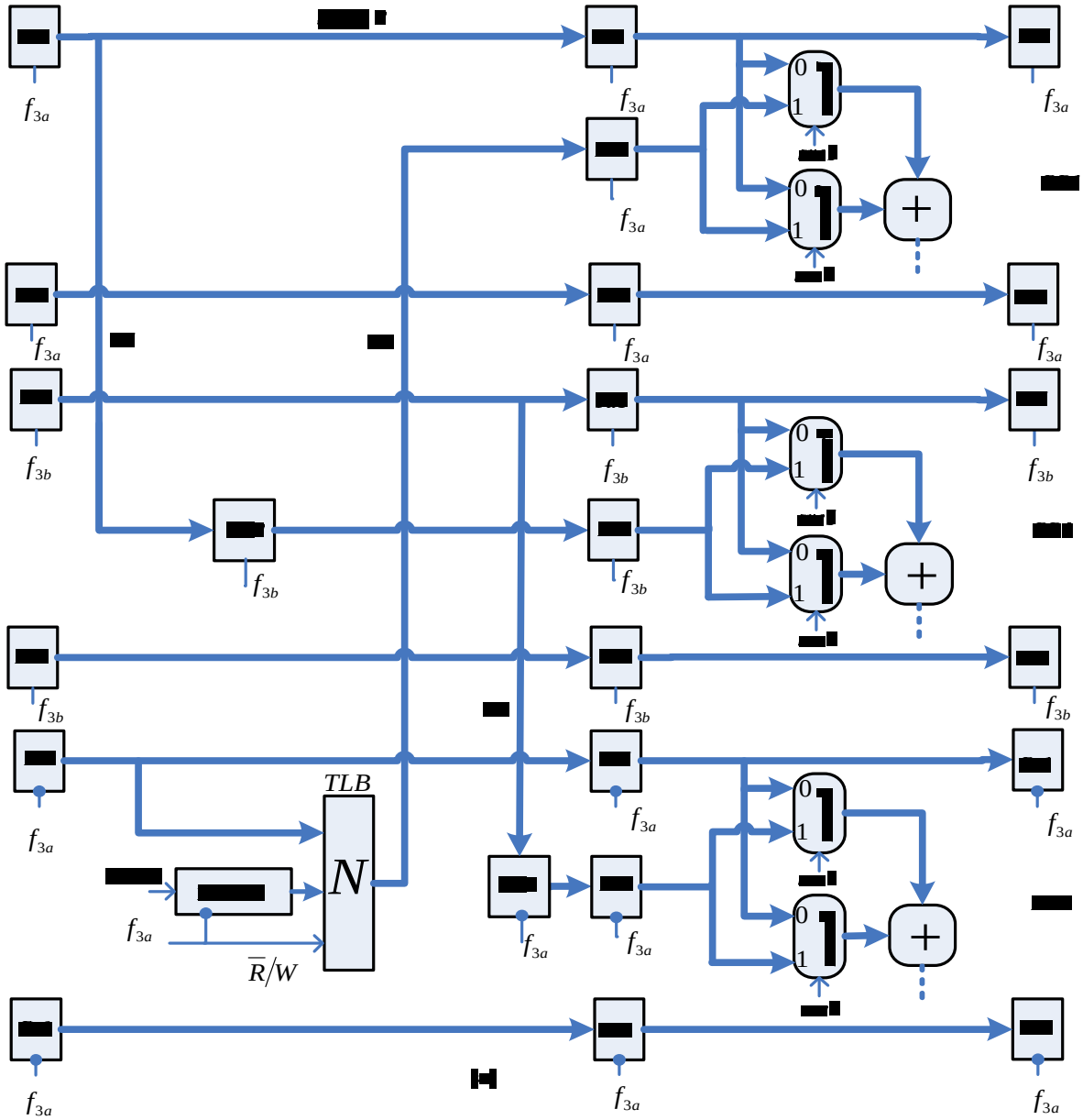
TABLE 8 DATAFLOW OF THE 3-PARALLEL INTERMEDIATE ARCHITECTURE

Ck	RP	Rd0	RP's input latches Rt0 Rt2 Rt1	RdH	SRH0 R2 R1 R0	SRH1 R2 R1 R0	RdL R1 R0	SRL0 R2 R1 R0	SRL1 R2 R1 R0
1	1	x 0,2	x 0,0 x 0,2 x 0,1						
2	2	x 0,4	x 0,2 x 0,4 x 0,3						
3	3	x 0,6	x 0,4 x 0,6 x 0,5						
4	1	x 1,2	x 1,0 x 1,2 x 1,1						
5	2	x 1,4	x 1,2 x 1,4 x 1,3						
6	3	x 1,6	x 1,4 x 1,6 x 1,5						
7	1	x 2,2	x 2,0 x 2,2 x 2,1						
8	2	x 2,4	x 2,2 x 2,4 x 2,3						
9	3	x 2,6	x 2,4 x 2,6 x 2,5						
10	1	x 3,2	x 3,0 x 3,2 x 3,1		H0,0 ----- -----			L0,0 ----- -----	
11	2	x 3,4	x 3,2 x 3,4 x 3,3		H0,1 H0,0 -----			L0,1 L0,0 -----	
12	3	x 3,6	x 3,4 x 3,6 x 3,5		H0,2 H0,1 H0,0			L0,2 L0,1 L0,0	
13	1	x 4,2	x 4,0 x 4,2 x 4,1		H0,2 H0,1 H0,0	H1,0 ----- -----		L0,2 L0,1 L0,0	L1,0 ----- -----
14	2	x 4,4	x 4,2 x 4,4 x 4,3		H2,0 H0,2 H0,1	H1,1 H1,0 -----		L2,0 L0,2 L0,1	L1,1 L1,0 -----
15	3	x 4,6	x 4,4 x 4,6 x 4,5		H0,2 H0,1 H0,0	H1,2 H1,1 H1,0		L0,2 L0,1 L0,0	L1,2 L1,1 L1,0
16	1	x 5,2	x 5,0 x 5,2 x 5,1		H2,0 H0,2 H0,1	----- H1,2 H1,1		L2,0 L0,2 L0,1	------ L1,2 L1,1
17	2	x 5,4	x 5,2 x 5,4 x 5,3	------	H2,1 H2,0 H 0,2	----- ------ H1,2	L2,1 -----	L2,0 L0,2 L0,1	------ L1,2 L1,1
18	3	x 5,6	x 5,4 x 5,6 x 5,5	H2,2	H2,1 H2,0 H 0,2	----- ------ H1,2	L2,2 L2,1	L2,0 L0,2 L0,1	------ L1,2 L1,1
19	1	x 6,2	x 6,0 x 6,2 x 6,1	------	H2,2 H2,1 H 2,0	H3,0 ----- ------	----- L2,2	L2,1 L2,0 L0,2	L3,0 ------ L1,2
20	2	x 6,4	x 6,2 x 6,4 x 6,3	------	H2,2 H2,1 H 2,0	H3,1 H3,0 ------	------ ------	L2,2 L2,1 L2,0	L3,1 L3,0 -----
21	3	x 6,6	x 6,4 x 6,6 x 6,5	------	H2,2 H2,1 H 2,0	H3,2 H3,1 H3,0	------ ------	L2,2 L2,1 L2,0	L3,2 L3,1 L3,0
22	1	x 7,2	x 7,0 x 7,2 x 7,1	------	H4,0 H2,2 H2,1	------ H3,2 H3,1	------ ------	L4,0 L2,2 L2,1	------ L3,2 L3,1
23	2	x 7,4	x 7,2 x 7,4 x 7,3	------	H4,1 H4,0 H2,2	------ ------ H3,2	L4,1 ------	L4,0 L2,2 L2,1	------ L3,2 L3,1
24	3	x 7,6	x 7,4 x 7,6 x 7,5	H4,2	H4,1 H4,0 H2,2	------ ------ H3,2	L4,2 L4,1	L4,0 L2,2 L2,1	------ L3,2 L3,1
25	1	x 8,2	x 8,0 x 8,2 x 8,1	------	H4,2 H4,1 H4,0	H5,0 ------ ------	----- L4,2	L4,1 L4,0 L2,2	L5,0 ----- L3,2
26	2	x 8,4	x 8,2 x 8,4 x 8,3	------	H4,2 H4,1 H4,0	H5,1 H5,0 ------	------ ------	L4,2 L4,1 L4,0	L5,1 L5,0 -----
27	3	x 8,6	x 8,4 x 8,6 x 8,5	-----	H4,2 H4,1 H4,0	H5,2 H5,1 H5,0	------ ------	L4,2 L4,1 L4,0	L5,2 L5,1 L5,0
28	1	x 9,2	x 9,0 x 9,2 x 9,1	------	H6,0 H4,2 H4,1	------ H5,2 H5,1	------ ------	L6,0 L4,2 L4,1	----- L5,2 L5,1
29	2	x 9,4	x 9,2 x 9,4 x 9,3	------	H6,1 H6,0 H4,2	------ ------ H5,2	L6,1 ------	L6,0 L4,2 L4,1	----- L5,2 L5,1
30	3	x 9,6	x 9,4 x 9,6 x 9,5	H6,2	H6,1 H6,0 H4,2	------ ------ H5,2	L6,2 L6,1	L6,0 L4,2 L4,1	----- L5,2 L5,1

ck	RP	CP1 & CP3 input latches Rt0 Rt2 Rt1 Rt0 Rt2 Rt1	CP2 input latches Rt0 Rt2 Rt1	CP1 & CP3 output latches Rth Rtl Rth Rtl	CP2 output latches Rth Rtl
16	1	H0,0 H2,0 H1,0 L0,0 L2,0 L1,0			
17	2	----------	H0,1 H2,1 H1,1		
18	3	----------	----------		
19	1	H0,2 H2,2 H1,2 L0,1 L2,1 L1,1	----------		
20	2	----------	L0,2 L2,2 L1,2		
21	3	----------	----------		
22	1	H2,0 H4,0 H3,0 L2,0 L4,0 L3,0	----------		
23	2	----------	H2,1 H4,1 H3,1		
24	3	----------	----------		
25	1	H2,2 H4,2 H3,2 L2,1 L4,1 L3,1	----------	HH0,0 HL0,0 LH0,0 LL0,0	
26	2	----------	L2,2 L4,2 L3,2	----------	HH0,1 HL0,1
27	3	----------	----------	----------	----------
28	1	H4,0 H6,0 H5,0 L4,0 L6,0 L5,0	----------	HH0,2 HL0,2 LH0,1 LL0,1	----------
29	2	----------		----------	LH0,2 LL0,2
30	3	----------	----------	----------	----------

in the path labeled *P3*. Then the positive transition of clock f_{3a} loads it into Rt2 in stage 3 of RP1 where it will be used in the calculation of the low coefficient. On the other hand, during the high pulse, as indicated in Fig. 9, the high coefficient that are needed in the next run will be stored in the TLB.

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 asserting *incAR*. Each time a run is complete, register TLBAR is reset zero to start a new run and the process is repeated.

The two multiplexers, labeled *muxh* and *muxl* in Fig. 8, are used for passing every clock cycle, with reference to clock f_3, the high and low coefficients, respectively, generated by the three RPs. The two control signals of the two multiplexers are shown in Fig. 8 connected to clocks f_{3a} and f_{3b}. When the two pulses of the clock f_{3a} and f_{3b} are low, the two multiplexers would pass the output coefficients generated by RP1, whereas when a high pulse of the clock f_{3a} and a low pulse of the clock f_{3b} occur, the two multiplexers would pass the output coefficients generated by RP2. Finally, when the two pulses are high, the two multiplexers would pass the output coefficient of RP3, as indicated in Fig. 9. In addition, note that the path extending from the inputs of the multiplexer *muxh*, passing through *muxh2*, *muxce0*, and ending at Rt2 may form a critical path, since signals through this path should reach Rt2 during one cycle of clock f_3.

The registers labeled SRH1, SRH0, SRL1, and SRL0, including RdH and RdL all operate with frequency f_3. Registers SRH1, SRH0, and RdH store high coefficients, while registers SRL1, SRL0, and RdL store low coefficients. First, according to Table 8, registers SRH0 and SRL0 each should be filled with 3 high and low coefficients, respectively. This process starts at clock cycle 10 and ends at cycle 12. Similarly, registers SRH1 and SRL1 each are filled with 3 high and low coefficients starting from cycles 13 to 15.

In cycle 16, CP1 latches are loaded with coefficients H0,0 and H1,0 transferred from SRH0 and SRH1, respectively, along with coefficient H2,0 generated by RP1 during the cycle. Note that coefficient H2,0 takes the path extending from the output of *muxh*, passing through *muxh2*, *muxce0* and

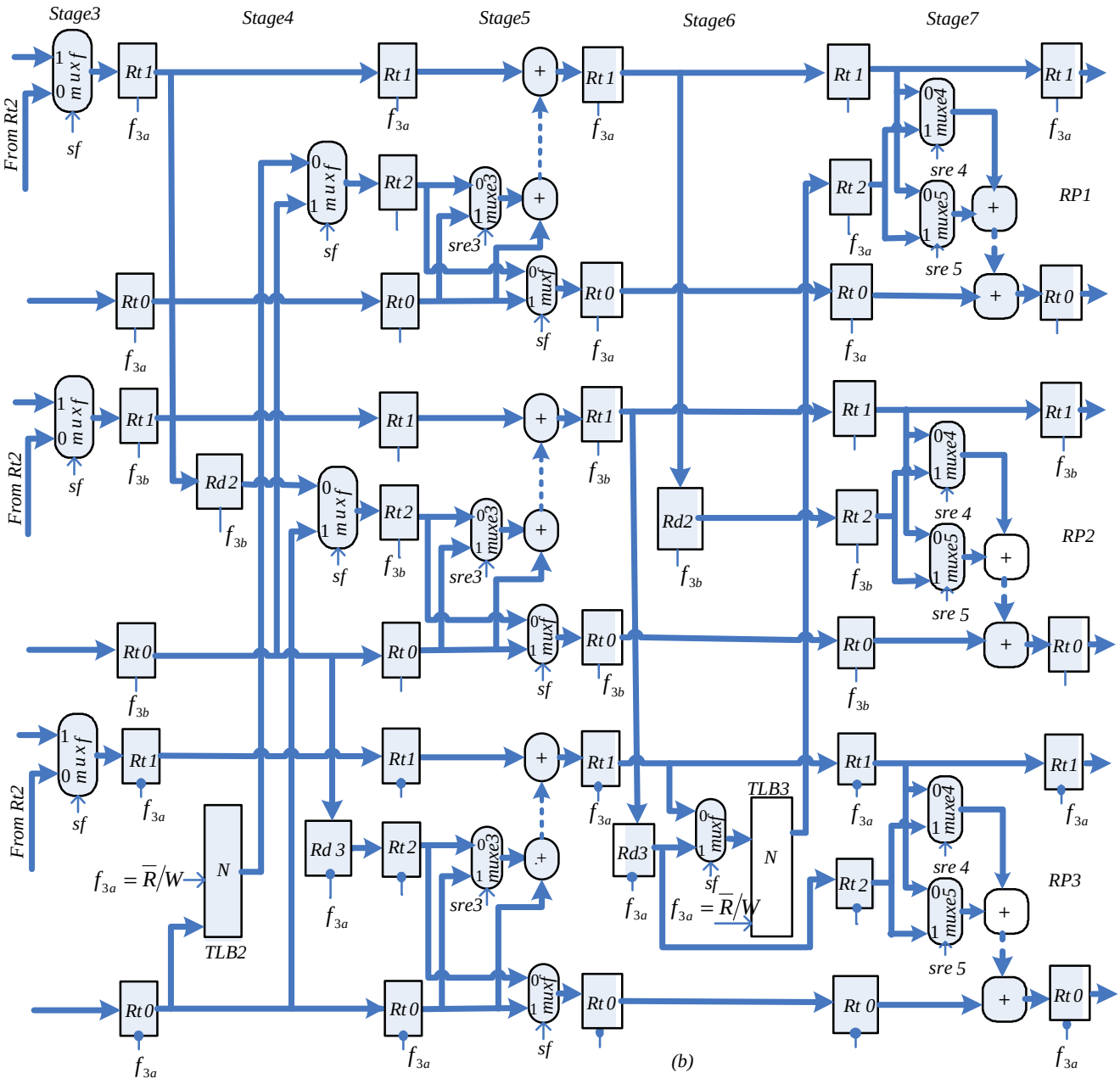


Fig. 10. (a, b) Modified 9/7 RPs datapath for 3-parallel intermediate architecture

ending at Rt2. Cycle 16 also loads CP3 latches with L0,0 and L1,0 transferred from SRL0 and SRL1, respectively, along with the coefficient L2,0 generated by RP1 during the cycle. In addition, during cycle 16, the coefficients H2,0 and L2,0 are also shifted into SRH0 and SRL0, respectively, as shown in Table 8. Cycle 17 loads, the two high coefficients H0,1 and H1,1 transferred from SRH0 and SRH1, respectively, along with the coefficient H2,1 generated by RP2 during the cycle, into CP2 latches. Cycle 17 also stores coefficients H2,1 and L2,1, generated by RP2 during the cycle, in SRH0 and RdL, respectively. The dataflow proceeds as shown in Table 8.

New coefficients are loaded simultaneously into both CP1 and CP3 latches every time clock f_{3a} makes a positive transition, whereas CP2 latches are loaded when clock f_{3b} makes a positive transition. Furthermore, each time a transition from a run to the next is made, when the column length *(N)* of an image is odd, the external memory should not be scanned for 3 clock cycles, since during this period the CPs will process the last high and low coefficients in each of the 3 columns of H and L decompositions, as required by the DDGs for odd signals. This is also true for 2-parallel intermediate architecture. No such situation occurs when the column length of an image is even.

From Table 8 it can be determined that the control signals of *muxh2*, *muxh3*, and register RdH can be combined into one signal, *sh2*. Similarly, the control signal of the two multiplexer's *muxl2*, *muxl3*, and register RdL can be combined into one signal, *sl2*. Furthermore, a careful examination of Table 8 shows that the control signal values that must be issued by the control unit for signals *sh1*, *sh0*, *sl1*, *sl0*, *sh2*, and *sl2*, starting from cycles 16 to 21 and repeat every 6 cycles, should be as shown in Table 9.

TABLE 9 CONTROL SIGNAL VALUES

Cycle	*sh1*	*sh0*	*sl1*	*sl0*	*sh2*	*sl2*
16	1	1	1	1	0	0
17	1	1	0	0	0	1
18	0	0	0	0	1	1
19	1	1	1	1	1	1
20	1	0	1	1	0	0
21	1	0	1	1	0	0

Finally, if it is necessary to extend the 2-parallel architecture to 4-parallel architecture, then from experience gained from designing 2- and 3-parallel architectures, the best architecture for 4-parallel would be obtained if the fourth overlapped scan method is used and 5-parallel if the fifth scan method is used and so on. Then the architecture design for a higher degree parallelism becomes similar to that experienced in the 3-parallel intermediate architecture. While an attempt, e.g., to design 4-parallel intermediate architecture using the third scan method would require very complex modifications in the datapath architecture of the combined 4 RPs and complex control logic. However, the objective for choosing a higher scan method in the first place is to reduce the power consumption due to overlapped areas scanning of external memory. Therefore, it makes sense if 4-parallel is designed with fourth scan method and 5-parallel with fifth scan method and so on.

IV. EVALUATION OF PERFORMANCE

To evaluate the performance of the two proposed parallel architectures in terms of speedup, throughput, and power consumption as compared with the single pipelined intermediate architecture proposed in [3] consider the following. In the single pipelined intermediate architecture [3], the total time, *T1*, required to yield *n* paired outputs for j-level decomposition of an *NxM* image is given by

$$\begin{aligned} T1 &= \left[\rho_1 + 3(n-1)\right]\tau_1 \\ T1 &= \left[\rho_1 + 3(n-1)\right]t_p/3k \end{aligned} \tag{5}$$

The dataflow of the 2-parallel architecture in Table 1 shows that $\rho_2 = 19$ clock cycles are needed to yield the first 2-pair of output. The remaining *(n-2)/2* outputs require *2(n-2)/2* cycles. Thus, the total time, *T2*, required to yield *n* paired outputs is given by

$$T2 = \left[\rho_2 + (n-2)\right]\tau_2$$

From (3) $\tau_2 = t_p/2k$ then

$$T2 = \left[\rho_2 + (n-2)\right]t_p/2k \tag{6}$$

The speedup factor *S* is then given by

$$S_2 = \frac{T1}{T2} = \frac{\left[\rho_1 + 3(n-1)\right]t_p/3k}{\left[\rho_2 + (n-2)\right]t_p/2k}$$

For large *n*, the above equation reduces to

$$S_2 = \frac{3(n-1)(2k)}{(n-2)(3k)} = 2 \tag{7}$$

Eq (7) implies that the proposed 2-parallel intermediate architecture is 2 times faster than the single pipelined intermediate architecture.

On the other hand, to estimate the total time, *T3*, required for j-level decomposition of an *NxM* image on the 3-parallel pipelined intermediate architecture, assume the output generated by CP2 in Table 8 are shifted up one clock cycle so that it parallel that of CP1 and CP3. Then, $\rho_2 = 25$ clock cycles are needed to yield the first 3-pair of output. The remaining *(n-3)/3* 3-paired outputs require *3(n-3)/3* clock cycles. Thus, the total time, *T3*, required to yield *n* paired outputs is given by

$$\begin{aligned} T3 &= \left[\rho_3 + (n-3)\right]\tau_3 \\ T3 &= \left[\rho_3 + (n-3)\right]t_p/3k \end{aligned} \tag{8}$$

The speedup factor *S* is then given by

$$S_3 = \frac{T1}{T3} = \frac{\left[\rho_1 + 3(n-1)\right]t_p/3k}{\left[\rho_3 + (n-3)\right]t_p/3k}$$

$$S_3 = \frac{3(n-1)}{(n-3)} = 3 \tag{9}$$

Eq (9) implies that the proposed 3-parallel pipelined intermediate architecture is 3 times faster than the single pipelined intermediate architecture.

The throughput, *H*, which can be defined as number of output coefficients generated per unit time, can be written for each architectures as

$$H(\sin gle) = n/\left(\rho_1 + 3(n-1)\right)t_p/3k$$

The maximum throughput, H^{max}, occurs when *n* is very large ($n \to \infty$). Thus,

$$\begin{aligned} H^{\max}(\sin gle) &= H(\sin gle)_{n\to\infty} \\ &\cong 3\cdot n\cdot k\cdot f_p/3\cdot n = k\cdot f_p \end{aligned} \tag{10}$$

$$H(2-parallel) = n/\left(\rho_2 + (n-2)\right)t_p/2k$$

$$\begin{aligned} H^{\max}(2-parallel) &= H(2-parallel)_{n\to\infty} \\ &\equiv 2\cdot n\cdot k\cdot f_p/n = 2\cdot k\cdot f_p \end{aligned} \tag{11}$$

$$H(3-parallel) = n/\left(\rho_3 + (n-3)\right)t_p/3k$$

$$\begin{aligned} H^{\max}(3-parallel) &= H(3-parallel)_{n\to\infty} \\ &\equiv 3\cdot n\cdot k\cdot f_p/n = 3\cdot k\cdot f_p \end{aligned} \tag{12}$$

Hence, the throughputs of the 2-parallel and 3-parallel pipelined architectures have increased by a factor of 2 and 3, respectively, as compared with single pipelined architecture.

To determine the amount of power reduction achieved in the external memory of the intermediate parallel architecture as compared with first scan method based parallel architecture, consider the following. The power consumption of VLSI architectures can be estimated as

$$P = C_{total}\cdot V_o^2\cdot f \tag{13}$$

where C_{total} denotes the total capacitance of the architecture, V_o is the supply voltage, and *f* is the clock frequency. Then the power consumption due to scanning external memory of single pipelined architecture based on nonoverlapped scan method [10] can be written as

$$P_s(non) = \beta\cdot C_{total}\cdot V_o^2\cdot f_1 \tag{14}$$

where $C_{total}\cdot V_o^2\cdot f_1$ is the external memory power consumption due to first overlapped scan method, f_1 is the

external memory scan frequency, and $\beta = T_{mn}/T_{mo} = 2/3$. T_{mo} and T_{mn} denote total external memory access time in clock cycle for *J* levels of decomposition for architecture based on the first overlapped and nonoverlapped scan methods, respectively, [3].

Using the fact that the scan method shown in Fig. 3 reduces the power consumption of the overlapped areas by a factor of 1/3, [3], the power consumption due to scanning the overlapped areas of Fig. 3 can be written as

$$P_o(areas) = \left(\beta_0 \cdot C_{total} \cdot V_o^2 \cdot f_1/3\right) \quad (15)$$

where $\beta_0 = T_{me}/T_{mo} = 1/3$ and T_{me} is the excess memory access time due to overlapped areas scanning for *J* levels of decomposition, [3].

Thus, the external memory power consumption of the single pipelined intermediate, $P_s(int)$, is

$$\begin{aligned} P_s(\text{int}) &= P_s(non) + P_o(areas) \\ &= \beta \cdot C_{total} \cdot V_o^2 \cdot f_1 + \beta_0 \cdot C_{total} \cdot V_o^2 \cdot f_1/3 \\ &= C_{total} \cdot V_o^2 \cdot f_1(\beta_0/3 + \beta) \\ &= 3 \cdot k \cdot C_{total} \cdot V_o^2 \cdot f_p(\beta_0/3 + \beta) \end{aligned} \quad (16)$$

where $f_1 = 3 \cdot k \cdot f_p$ [10], and f_p is processor's frequency.

The power consumption of l-parallel pipelined intermediate architecture, $P_l(\text{int})$ can be written as

$$P_l(\text{int}) = I \cdot C_{total} \cdot V_o^2 \cdot f_l \cdot (\beta_0/3 + \beta)$$

From (3) $1/\tau_l = f_l = l \cdot k/t_p$, then

$$P_l(\text{int}) = l \cdot k \cdot I \cdot C_{total} \cdot V_o^2 \cdot f_p \cdot (\beta_0/3 + \beta) \quad (17)$$

where *I* is number of input buses and it is 3 in the parallel architecture.

Similarly, the external memory power consumption of l-parallel pipelined architecture based on the first scan, $P_L(first)$ can be written as

$$\begin{aligned} P_l(first) &= I \cdot C_{total} \cdot V_o^2 \cdot f_l \\ &= l \cdot k \cdot I \cdot C_{total} \cdot V_o^2 \cdot f_p \end{aligned} \quad (18)$$

Thus,

$$\frac{P_l(\text{int})}{P_l(first)} = \frac{l \cdot k \cdot I \cdot C_{total} \cdot V_o^2 \cdot f_p \cdot (\beta_0/3 + \beta)}{l \cdot k \cdot I \cdot C_{total} \cdot V_o^2 \cdot f_p} = \beta_0/3 + \beta = 7/9$$

implies that the intermediate parallel architecture based on scan method in Fig. 3 reduces power consumption of the external memory by a factor of 7/9 as compared with parallel architecture based on the first scan method. While,

$$\frac{P_l(\text{int})}{P_s(\text{int})} = \frac{l \cdot k \cdot I \cdot C_{total} \cdot V_o^2 \cdot f_p \cdot (\beta_0/3 + \beta)}{3 \cdot k \cdot C_{total} \cdot V_o^2 \cdot f_p \cdot (\beta_0/3 + \beta)} = l$$

implies that as the degree of parallelism increases the external memory power consumption of the intermediate parallel architecture based on the scan method in Fig. 3 also increases by a factor of l as compared with single pipelined intermediate architecture's external memory power consumption.

V. SCALE FACTOR MULTIPLIERS REDUCTION

In the lifting-based tree-structured filter bank for 2-D DWT shown in Fig. 11, it can be observed that the high output coefficients, which form H decomposition, each is multiplied by the scale factor *k* in the first pass. In the second pass, the high output coefficients, which form HH subband, each is multiplied by *k*. This implies the first multiplication can be eliminated and the output coefficients of HH subband can be multiplied by k^2 using one multiplier after the second pass. While, the high output coefficients, which form HL subband, each is multiplied by 1/*k*. This implies no multiplications are required and scale multipliers along this path can be eliminated, since HL subband coefficients are formed by multiplying each coefficient in the first pass by *k* and then in the second by pass by 1/*k*.

On the other hand, the low output coefficients of the first pass, which form L decomposition, each is multiplied by 1/*k*. Then in the second pass, the output coefficients, which form LH subband, each is multiplied by *k*, which implies no multiplications are required along this path. While, the output coefficients of the second pass, which form LL subband, each is multiplied by 1/*k*. Thus, instead of performing two multiplications, one multiplication can be performed by $1/k^2$ after the second pass [5, 7, 9]. However, note that the simple computations involve in each lifting step shown in Eqs (1) and (2) have made arriving at these results possible.

This process reduces number of multipliers used for scale factor multiplications in the tree-structured filter bank to 2 instead of 6 multipliers. When it applied to single pipelined architectures, it reduces number of scale multipliers to 2 instead of 4, whereas, in 2- and 3-parallel pipelined architectures, it reduces number of scale multipliers to 2 and 4 instead of 8 and 12, respectively.

In [14], it has been illustrated that the multipliers used for scale factor *k* and coefficients $\alpha, \beta, \gamma,$ and δ of the 9/7 filter can be implemented in hardware using only two adders.

VI. COMPARISONS

Table 10 shows the comparison results of the proposed architectures with most recent architectures in the literature. In [12], a new method was introduced to shorten the critical path delay of the lifting-based architecture to one multiplier delay but requires a total line buffer of size 4*N*, which is a very expensive memory component, while the two proposed architectures each requires only a total line buffer of size 3*N*. In [8, 11], by reordering the lifting-based DWT of the 9/7 filter, the critical path of the pipelined architectures have been reduced to one multiplier delay but requires a total line buffer of size 5.5*N*. The architecture proposed in [4], achieves a critical path of one multiplier delay using very large number of pipeline registers. In addition, it requires a total line buffer of size 6*N*. In the efficient pipelined architecture [9], a critical path delay of Tm+Ta is achieved through optimized data flow

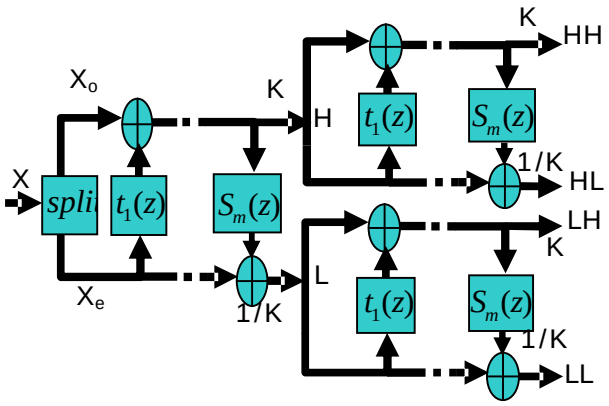


Fig. 11. Lifting-based tree-structured filter bank

graph but requires a total line buffer of size 10*N*.

On the other hand, the architectures proposed in [13, 7], like the proposed 2-parallel intermediate architecture, achieves a speedup factor of 2 as compared with other architectures. However, [13], the deeply parallel architecture requires a total line buffer of size 5*N*, whereas [7] requires a total line buffer of size 5.5*N*.

The main objective for introducing intermediate architecture is to reduce the external memory power consumption, which consumes the most power, while in other architectures this is not a concern. In addition, our architectures as compared with architectures listed in Table 10 are accurate, complete, and more practical.

TABLE 10 COMPARISONS OF SEVERAL 1-LEVEL (9/7) 2-D DWT ARCHITECTURES

Architecture	Multi	Adders	Line Buffer	Computing Time	Critical Path
Chao [11]	6	8	5.5N	$2(1-4^{-j})$NM	Tm
PLSA [12]	12	16	4N	N/A	Tm
Bing [8]	6	8	5.5N	$2(1-4^{-j})$NM	Tm
Lan [4]	12	12	6N	$2(1-4^{-j})$NM	Tm
Overlapped[10]	10	16	3N	$2(1-4^{-j})$NM	Tm + 2Ta
Jain [9]	9	16	10N	$2(1-4^{-j})$NM	Tm + Ta
Cheng [7]	18	32	5.5N	$(1-4^{-j})$NM	N/A
FIDF [13]	24	32	5N	$(1-4^{-j})$NM	Tm + 2Ta
Proposed (2-parallel)	18	32	3N	$(1-4^{-j})$NM	Tm + 2Ta
Proposed (3-parallel)	28	48	3N	$2/3(1-4^{-j})$NM	Tm + 2Ta

Tm: multiplier delay Ta: adder delay

VII. CONCLUSIONS

In this paper, two highly efficient parallel VLSI architectures for the 5/3 and the 9/7 algorithms that achieve speedup factor of 2 and 3 as compared with the single pipelined intermediate architecture are proposed. The advantage of the two proposed architectures is that each requires only a total line buffer of size *N* in the 5/3 processors and *3N* in the 9/7 processors. While other architectures in Table 10 require more line buffers, which are very expensive memory components. The two proposed parallel architectures could be very excellent candidates in real-time applications of 2-D DWT, where very high-speed, low power, and low cost are required. In addition, a control circuit diagram for determining the last run, which needs especial treatment, is provided. Furthermore, to reduce control design afford we have provided also several tables giving the control signal values for several control signals.

REFERENCES

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

[2] W. Sweldens, "The lifting scheme: A new philosophy in biorthogonal wavelet constructions," proc. SPIE 2569, 1995, PP. 68-79.

[3] Ibrahim 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.

[4] 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.

[5] S. Mowa, S. Srinivasan, "A novel architecture for lifting-based discrete wavelet transform for JPEG2000 standard suitable for VLSI implementation," Proceedings of the 16th International Conf. on VLSI Design, 2003 IEEE, PP. 202 – 207.

[6] 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.

[7] 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.

[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 JPEG2000 Codec," IEEE Trans. on Circuits & Sys. for Video Technology, Vol. 15, No. 12, December 2005, PP. 1615 – 1628.

[9] R. Jain and P. R. Panda,"An efficient pipelined VLSI architecture for Lifting-based 2D-discrete wavelet transform," ISCAS, 2007 IEEE, PP. 1377-1380.

[10] I. 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.

[11] W. Chao, W. Zhilin, C. Peng, and L. Jie, "An efficient VLSI Architecture for lifting-based discrete wavelet transform," Mulltimedia and Epo, 2007 IEEE International conference, PP. 1575-1578.

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

[13] B-F. Li and Y. Dou, "FIDP A novel architecture for lifting-based 2D DWT in JPEG2000," MMM (2), lecture note in computer science, vol. 4352, PP. 373-382, Springer, 2007.

[14] Qing-ming Yi and Sheng-Li Xie,"Arithmetic shift method suitable for VLSI implementation to CDF 9/7 discrete wavelet transform based on lifting scheme," Proceedings of the Fourth Int. Conf. on Machine Learning and Cybernetics, Guangzhou, August 2005, PP. 5241-5244.