Configurable VLSI Architecture for Deblocking Filter in H.264/AVC

Chung-Ming Chen and Chung-Ho Chen, *Member, IEEE*

***Abstract*—In this paper, we study and analyze the computational complexity of the deblocking filter in H.264/AVC baseline decoder based on SimpleScalar/ARM simulator. The simulation result shows that the memory reference, content activity check operations, and filter operations are known to be very time consuming in the decoder of this new video coding standard. In order to improve overall system performance, we propose a configurable, extensible, and synthesizable window-based processing architecture which simultaneously processes the horizontal filtering of vertical edge and vertical filtering of horizontal edge. As a result, the memory performance of the proposed architecture is improved by four times when compared to previous designs. Moreover, the system performance of our window-based architecture significantly outperforms the previous designs from 7 times to 20 times.**

***Index Terms*—Deblocking filter, H.264/AVC, video coding.**

I. Introduction

VIDEO compression is the critical technology in today's multimedia systems. The limited transmission bandwidth or storage capacity for applications such as HDTV, video conferencing, 3G for mobile device, and Internet video streaming emphasizes the demand for higher video compression rates. To achieve this demand, the new video coding standard Recommendation H.264 of ITU-T [1], also known as International Standard 14496-10 or MPEG-4 Part 10 Advanced Video Coding (AVC) of ISO/IEC, has been developed. It significantly outperforms the previous ones (i.e., H.261 [2], MPEG-1 Video [3], MPEG-2 Video [4], H.263 [5], and MPEG-4 Visual or part 2 [6]) in bit-rate reduction. The functional blocks of H.264/AVC, as well as their features, are shown in Fig. 1. Comparing the H.264/AVC video coding tools (e.g., adaptive deblocking filter [7], integer DCT-like transform [8] instead of the DCT [9], multiple reference frame [10], new frame types (SP-frames and SI-frames) [11], further predictions using B-slices [12], quarter per motion compensation [13], or CABAC [14]) to the tools of previous video coding standard, H.264/AVC provides the most improved algorithm in the evolution of video coding as well as error robustness and network friendliness [15]–[20]. At the same time, preliminary studies [21] using software based on this new standard suggest that H.264 offers up to 50% better compression than MPEG-2 and up to 30% better than H.263+ and MPEG-4 advanced simple profile.

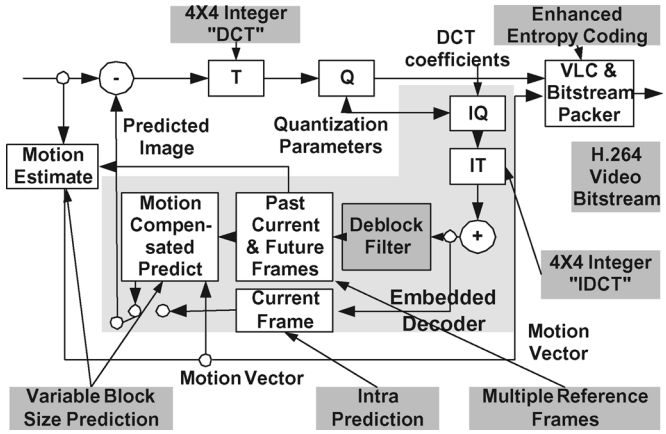

Fig. 1. Block diagram of H.264/AVC.

As our experimental result indicates, the operation of the deblocking filter is the most time consuming part of H.264/AVC video decoder. The block-based structure of the H.264/AVC architecture produces artifacts known as blocking artifacts. These blocking artifacts can occur from both quantization of the transform coefficients and block-based motion compensation. In order to reduce the blocking artifacts, the overlapped block motion compensation (OBMC) [22] is adopted into H.263 standard. Unlike the OBMC in H.263, H.264/AVC adopts an adaptive deblocking filter [7] that has shown to be a more powerful tool in reducing artifacts and improving the video quality. As a result, the filter reduces the bit rate typically by 6%–9% while producing the same objective quality as the nonfiltered video [23]. As shown in Fig. 2(a) a nonfiltered and Fig. 2(b) a filtered image, we can observe the different image quality between the nonfiltered and the filtered image at QP equal to 36. Adaptive deblocking filter can also be used in interpicture prediction to improve the ability to predict other picture as well. Since it is within the motion compensation prediction loop, the deblocking filter is often referred to as an in-loop filter. A detailed description of the adaptive deblocking filter can be found in [7].

The filtering operations of H.264/AVC standard require more instructions to process deblocking. Due to intensive computations, in [24]–[33] and [34] dedicated hardware was developed for acceleration. However, the deblocking filter described in the H.264/AVC standard is highly adaptive. Several parameters and thresholds, as well as the content of the picture itself, control the boundary strength of the filtering process. These issues are also equally challenging during parallel processing under

Manuscript received May 10, 2006; revised March 31, 2007 and April 2, 2007. This work was supported by the National Science Council, Taiwan, under NSC Contract 94-2220-E-006-004.

C.-M. Chen is with the Electrical Engineering Department, National Cheng Kung University, Tainan City 701, Taiwan, R.O.C. (e-mail: cmchen@ee.ncku.edu.tw).

C.-H. Chen is with the Electrical Engineering Department and Institute of Computer and Communication Engineering, National Cheng Kung University, Tainan City 701, Taiwan, R.O.C. (e-mail: chchen@mail.ncku.edu.tw).

Digital Object Identifier 10.1109/TVLSI.2008.2000516

(a) (b)

Fig. 2. Quality between nonfiltered and filtered images. (a) Unfiltered (QP = 36). (b) Filtered (QP = 36).

DSP or SIMD computational architecture. In order to reduce the conditional branch operations, we include the content activity check operations, table-derived operations, filtering operations, and computation of boundary strength operations into the edge filtering operation (EFO) unit to accelerate the deblocking filtering of H.264/AVC video coding. In addition, we propose an efficient VLSI architecture to improve memory performance by four times when compared to previous designs. The proposed architecture is called configurable window-based architecture (WIN). It uses a novel processing order within a frame to simultaneously process the horizontal filtering of vertical edge and vertical filtering of horizontal edge. Hence, our architecture is able to significantly improve the system performance and reduce the power consumption in the embedded system.

The organization of this paper is as follows. In Section II, the algorithm of the deblocking filter is given. Section III analyzes the computational complexity of H.264/AVC baseline decoder. Section IV illustrates the various configuration of our proposed architecture. Section V shows the simulation results. Finally, the conclusion is presented in Section VI.

II. Algorithm of Deblocking Filter

In this section, we briefly describe the algorithm of deblocking filter in H.264/AVC from processing order to sample processing level. A detailed description of the adaptive deblocking filter can be found in [7].

A. Processing Order

As the H.264/AVC standard suggests [1], for each luminance macroblock, the left-most edge of the macroblock is filtered first, followed by the other three internal vertical edges from left to right. Similarly, the top edge of the macroblock is filtered first, followed by the other three internal horizontal edges from top to bottom. Chrominance filtering follows a similar order in each direction for each 8×8 chrominance macroblock, as shown in Fig. 3.

According to this rule, there are four types of processing orders, which are proposed by [24], [26], and [28], as shown in Figs. 4–7. It is obvious that adaptive deblocking filter shall be applied to all 4×4 block edges of a picture, except for the edges at the boundary of the picture. Therefore, most of the 4×4 blocks need to be filtered four times with the adjacent blocks (left, right, top, and bottom). In order to improve the memory performance, we propose a configurable and extensible VLSI architecture with novel processing order to reduce memory reference of each 4×4-block to one as will be presented in Section IV.

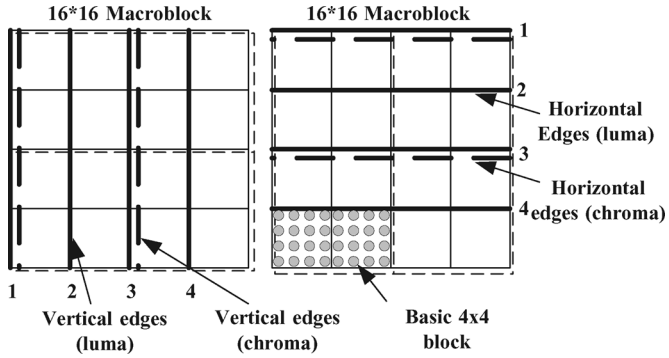

Fig. 3. Processing order depicted in standard.

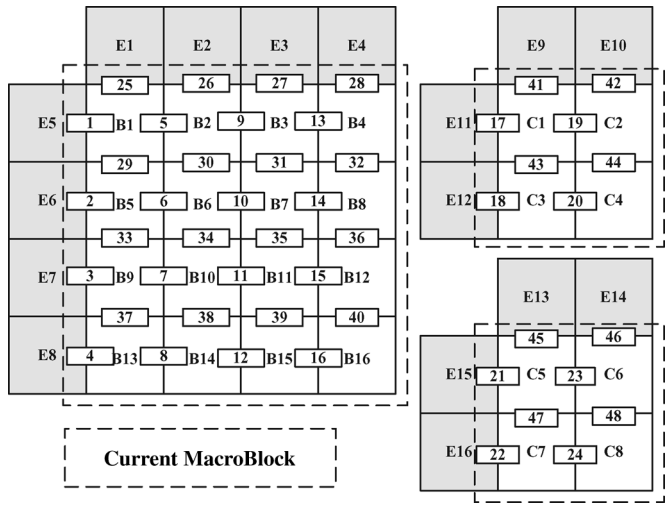

Fig. 4. Basic processing order of [24].

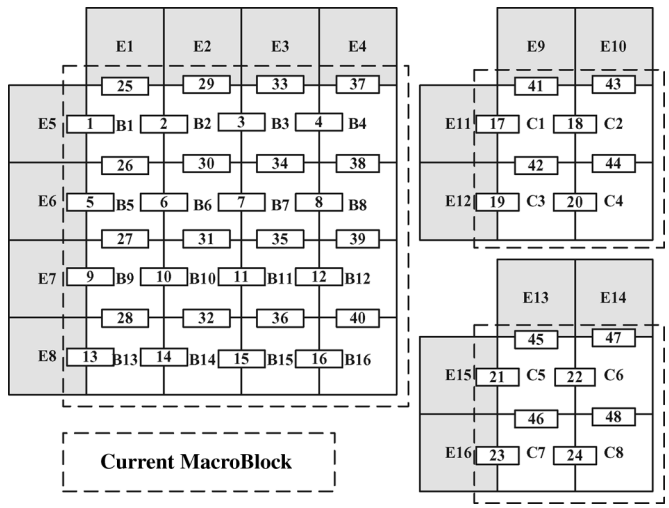

Fig. 5. Advanced processing order of [24].

B. Sample Processing Level

On the sample processing level, the quantization parameter, threshold value of Alpha and Beta, and content of the picture itself can turn on or turn off the filtering for each individual set of sample. For example, Fig. 8 illustrates the principle of the deblocking filter using a one-dimensional visualization of

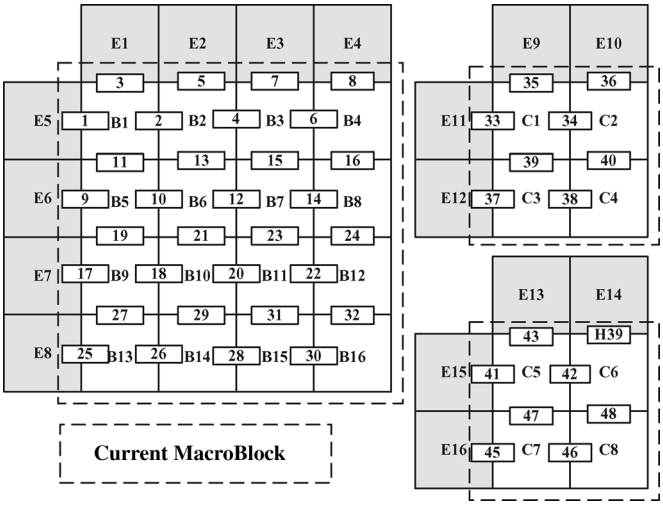

Fig. 6. Processing order of [26].

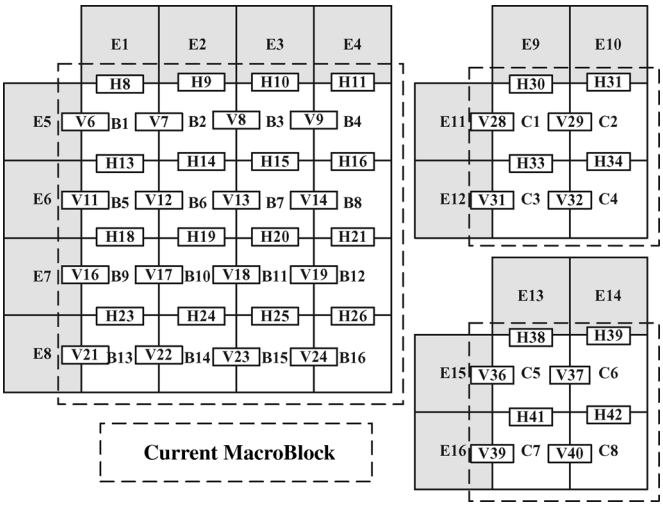

Fig. 7. Novel processing order of [28].

a block edge in a typical situation where the filter would be turned on. Whether the samples p_0 and q_0 as well as p_1 and q_1 are filtered is determined by using Boundary Strength (Bs), dependent threshold Alpha(QP) and Beta(QP), and content of the picture itself. Thus the filtering of p_0 and q_0 only takes place if the following content activity check operations are satisfied:

$$\text{Bs!} = 0 \tag{1}$$

$$|p_0 - q_0| < \text{Alpha(QP)} \tag{2}$$

$$|p_1 - p_0| < \text{Beta(QP)} \text{ and } |q_1 - q_0| < \text{Beta(QP)}. \tag{3}$$

Correspondingly, the filtering of p_1 or q_1 takes place if the condition below is satisfied:

$$|p_2 - p_0| < \text{Beta(QP)} \text{ and } |q_2 - q_0| < \text{Beta(QP)}. \tag{4}$$

The dependency of Alpha and Beta on the quantizer links the strength of filtering to general quality of the reconstructed picture prior to filtering. For small quantizer values, the thresholds

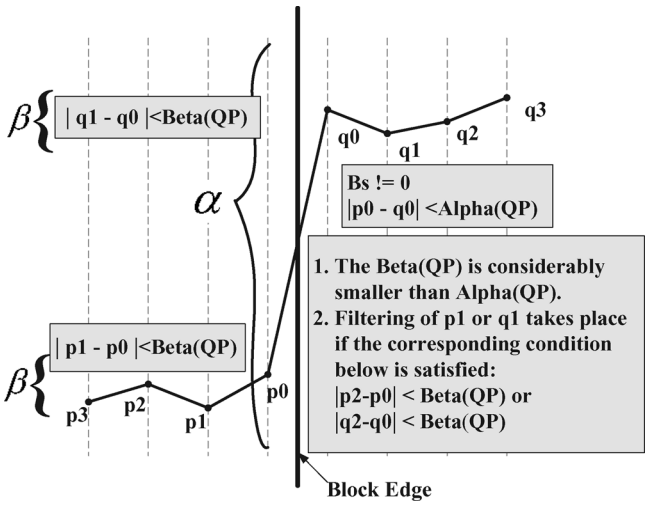

Fig. 8. Principle of deblocking filter.

TABLE I
SIMULATOR PARAMETER

Parameter	value
Fetch Queue size	4
Fetch Speed	1
Decode Width	1
Issue Width	1
Commit Width	1
D-Cache	32-way, LRU, 1-cycle hit, total 16KB
I-Cache	32-way, LRU, 1-cycle hit, total 8KB
Memory Latency	12
Memory Width	4 bytes

both become zero, and filtering is effectively turned off altogether.

The basic idea is that if a relatively large absolute difference between samples near a block edge is measured, it is quite likely to be a blocking artifact and should therefore be reduced. However, if the magnitude of that difference is so large that it can no longer be explained by the coarseness of the quantization used in the encoding, the edge is more likely to reflect the actual behavior of the source picture and should not be smoothed over.

III. COMPUTATIONAL COMPLEXITY

The simulator used in this study is derived from the SimpleScalar/ARM tool set [35], a suite of functional and timing simulation tools for ARM ISA. Our baseline simulation configuration models the Intel's StrongARM SA-110 processor. The hardware parameter is listed in Table I.

The H.264/AVC JM9.2 code [36] is used for reporting the complexity assessment experiments. The test sequences used in the computational complexity assessment are Forman 30 Hz QCIF and CIF, Mother and Daughter 30 Hz QCIF, and Mobile and Akiyo 30 Hz CIF. A fixed quantization parameter setting with a QP of 28 has been assumed.

One of the most important issues in the computational complexity of the H.264/AVC decoder is the distribution of time complexity among its major subfunction. In our simulation result shown in Table II, deblocking filtering (36%) is the largest component, followed by interpolation (22%), bitstream parsing

TABLE II
COMPUTATIONAL COMPLEXITY OF DECODER

Function	Complexity
Deblocking Filtering	36%
Interpolation	22%
Entropy Coding	13%
Inverse Transfers and Reconstruction	13%
Others	16%

and entropy decoding (13%), and inverse transfers and reconstruction (13%).

IV. PROPOSED ARCHITECTURE

In this paper, our proposed architecture can be configured as five types of processing engine. Configuration 1 is the edge filtering unit, which includes content activity check operations, table-derived operations, and the edge filter unit. Configuration 2 is the SPA, a simultaneous process engine that employs a novel processing order to simultaneously process the horizontal filtering of vertical edge and vertical filtering of horizontal edge. Configuration 3 is the vertical processing approach, which uses buffering scheme to reuse previous loaded block and the reduce the number of memory references. Configuration 4, is the SPA with a parallel processing engine to improve the overall performance. Finally, Configuration 5 is the configurable and extensible window-based architecture with parallel processing engine that uses memory FIFO instead of register file to reduce system cost.

A. Configuration 1: Edge Filtering Operation Unit

The complexity of the H.264/AVC Deblocking Filter mainly comes from two places. The first is the high adaptive filtering, which requires several conditional processing on each block edge and sample level. As described in the previous section, the computation of boundary strength, the threshold value of Alpha and Beta, the table-derived operations, and EFO are known to be very time consuming. Therefore, we propose an efficient VLSI architecture that includes these content activity check operations into the EFO unit to accelerate the horizontal and vertical filtering on the boundary of two adjacent basic 4×4 blocks, as shown in Fig. 9. A detailed description of the edge filtering unit can be found in [27]. There are five subfunctions in our EFO unit, as described below.

1) Computation of Boundary Strength: The purpose of computing boundary strength is to determine whether a block artifact may have been produced across the boundary, and thus determine the appropriate strength (Bs) of the filter to be used on the edge. A Boundary Strength (Bs) is assigned an integer value from 0 to 4. The strongest filter $(\mathrm{Bs} = 4)$ is used if one or both sides of edges are intra coded and the boundary is a macroblock boundary, whereas a value of 0 means no filtering is applied on this specific edge. In the standard mode of filtering which is applied for edges with Bs from 1 to 3, the value of Bs affects the maximum modification of the sample values that can be caused by filtering. Fig. 10 shows how the value of Bs is determined.

2) Filtering Operation: The most important function of the deblocking filter is the filtering operation, which is divided into two modes. A special mode of filtering that allows for stronger

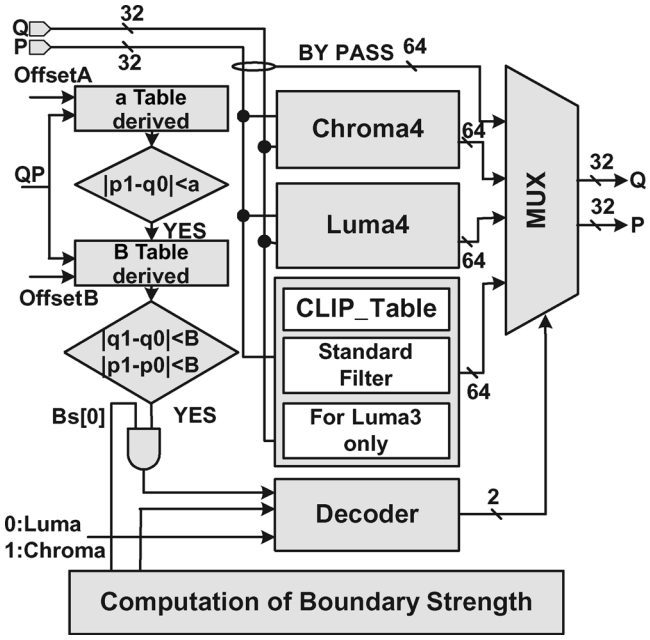

Fig. 9. EFO unit.

filtering is applied when Bs is equal to 4. The others are standard mode of filtering with a Bs parameter of 1 to 3, as shown in the following.

Luma4 for Q sample: $(\mathbf{Bs} = 4)$

$$q_0' = (p_1 + 2p_0 + 2q_0 + 2q_1 + q_2 + 4) \gg 3 \quad (5)$$
$$q_1' = (p_0 + q_0 + q_1 + q_2 + 2) \gg 2 \quad (6)$$
$$q_2' = (2q_3 + 3q_2 + q_1 + q_0 + p_0 + 4) \gg 3 \quad (7)$$

Luma4 for P sample: $(\mathbf{Bs} = 4)$

$$p_0' = (q_1 + 2q_0 + 2p_0 + 2p_1 + p_2 + 4) \gg 3 \quad (8)$$
$$p_1' = (q_0 + p_0 + p_1 + p_2 + 2) \gg 2 \quad (9)$$
$$p_2' = (2p_3 + 3p_2 + p_1 + p_0 + q_0 + 4) \gg 3 \quad (10)$$

Chroma4 for P and Q sample: $(\mathbf{Bs} = 4)$

$$q_0' = (2q_1 + q_0 + p_1 + 2) \gg 2 \quad (11)$$
$$p_0' = (2p_1 + p_0 + q_1 + 2) \gg 2 \quad (12)$$

Luma and Chrom: **(Bs = 3, 2, and 1)**

$$\mathrm{Dif} = \mathrm{Clip}\,(-t_c, t_c, ((((q_0 - p_0) \ll 2) + (p_1 - q_1) + 4) \gg 3)) \quad (13)$$
$$p_0' = \mathrm{Clip}(p_0 + \mathrm{Dif}) \quad (14)$$
$$q_0' = \mathrm{Clip}(q_0 - \mathrm{Dif}) \quad (15)$$

Luma only: **(Bs = 3, 2, and 1)**

$$\mathrm{Dif} = \mathrm{Clip}\,(-t_{c0}, t_{c0}, (p_2 + ((p_0 + q_0 + 1) \gg 1) - (p_1 \ll 1)) \gg 1) \quad (16)$$
$$p_1' = p_1 + \mathrm{Dif} \quad (17)$$
$$\mathrm{Dif} = \mathrm{Clip}\,(-t_{c0}, t_{c0}, (q_2 + ((p_0 + q_0 + 1) \gg 1) - (q_1 \ll 1)) \gg 1) \quad (18)$$
$$q_1' = q_1 + \mathrm{Dif} \quad (19)$$

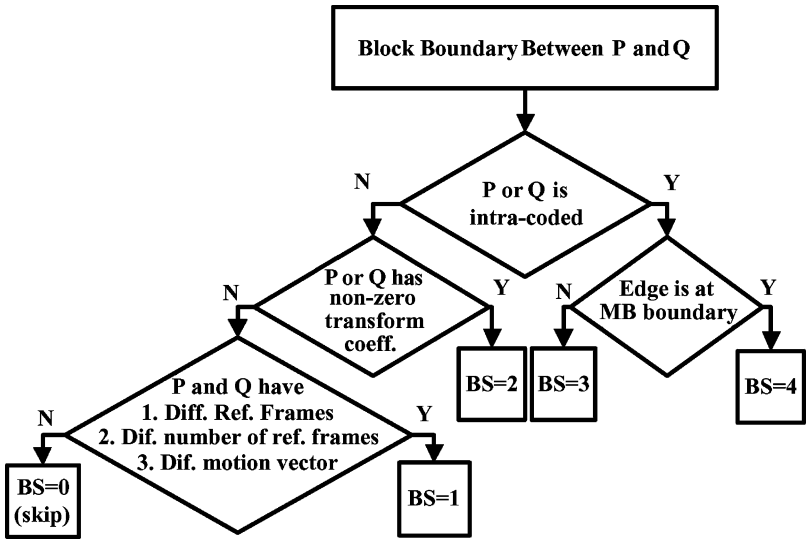

Fig. 10. Flowchart for determining boundary strength.

3) Clipping Operation: The filtering operation may result in too much low-pass filtering (blurring). A significant part of the adaptive filter is obtained by limiting these values. This process is called clipping. There are five clipping operations in our proposed architecture, as shown in the filtering operation. A detailed description of the clipping operation can be found in [7].

4) Content Activity Check Operation: Conditional branches which are described below almost inevitably appear in the inner most loops of the algorithm. The major content activity checks (conditional branches) are listed in (1)to (4) and described in Section II.

5) Table-Derived Operations: In order to simultaneously access Alpha, Beta, and Clip tables, we used combinational logic to implement Alpha, Beta, and Clip tables instead of using memory buffer, since most values of these tables are zero. It can save most of the space of memory buffer, improve overall system performance, and reduce power consumption.

B. Configuration 2: Simultaneous Processing Architecture

Another reason for the high complexity is the small block size employed for residual coding in the H.264/AVC video coding algorithm. With the 4×4 blocks and a typical filter length of two samples in each direction, each sample in a picture must be loaded/stored from/to memory four times either to be modified or to determine if the neighboring samples will be modified. In order to reduce the number of memory references and improve the overall system performance, we propose the second configuration architecture, which can simultaneously process the horizontal filtering of vertical edge and vertical filtering of horizontal edge, as shown in Fig. 11. The proposed architecture is called simultaneous processing architecture (SPA).

There are three major subfunctions in our SPA architecture. The first component is the Shift Operation Array (SOA). There are six forwarding shift register arrays in our proposed architecture (for example, A1, W1, W2, W3, W4, and W5). Each array contains four entries which contains four processed samples. The shift direction is shown in Fig. 11. The second function in our proposed architecture is the transposing operation (i.e., T1 and T2), as shown in Fig. 11. T1 and T2 latch the 4×4 block sample values that are transposed from A1 and W5, respectively. The final important functions are the horizontal and vertical filter units, which are described in the previous subsection.

The basic macroblock processing sequence using raster scan order and the number of block cycles for processing each macroblock of a QCIF frame are shown in Fig. 12. Table III and Fig. 7 presents the data flow of processing a macroblock. For a basic 4×4-block, it takes 13 block cycles (52 clock cycles) to process the first block B1 and the number of total processing time for each 16×16 macroblock is 32 block cycles (128 clock cycles). In the first 5 block cycles (the first 20 clock cycles), the blocks E1, E2, E3, E4, and E5 are loaded from the internal memory to SPAs W3, W2, W1, T1, and A1, respectively, and no filtering operations are performed. In the next two block cycles (state 6 and state 7 in Table III), the horizontal filtering of vertical edge V6 and V7 are performed sequentially. At the eighth block cycle, the proposed architecture SPA can simultaneously process the horizontal filtering of vertical edge V8 (the boundary of block B2 and B3) and the vertical filtering of horizontal edge H8 (the boundary of block E1 and B1), and write block E1 to the internal memory at the next block cycle (the ninth block cycle). At the thirteenth block cycle, the vertical filtering of horizontal edge H13 (the boundary of block B1 and B5) is performed. At this time, block B1 has finished 4 times of filtering with the adjacent blocks (left block E5, right block B2, top block E1, and bottom block B5). Finally, SPA writes block B1 to the internal memory at the fourteenth block cycle. Therefore, the number of total processing time for a QCIF frame is 5537 block cycles (Luma is 2863, two Chroma are $1337 \times 2 = 2674$).

C. Configuration 3: Vertical Processing Architecture

The basic idea of the vertical processing approach is to save the initial phase (seven or eight block cycles) and buffer the initial blocks B13, B14, B15, and B16 in SPAs W2, W1, T1, and A1, respectively, as shown in Fig. 13 and Table IV. When processing the next macroblock M2, the SPA with vertical processing order does not need to process the initial step nor load these initial blocks. Thus we can save seven (cluster 1) or eight

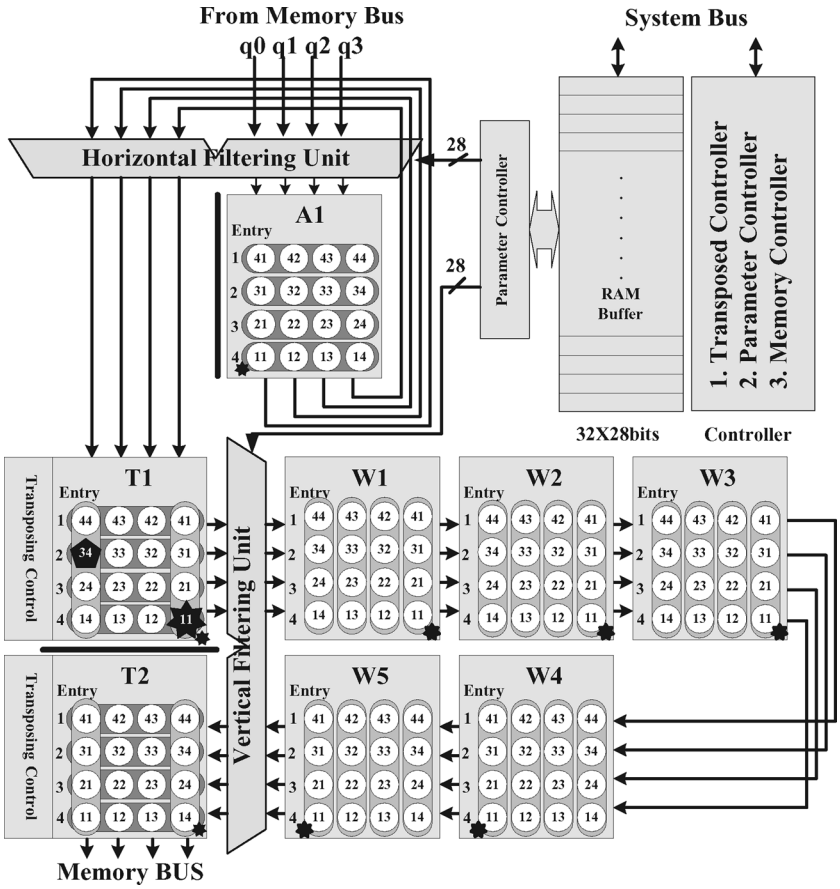

Fig. 11. SPA architecture.

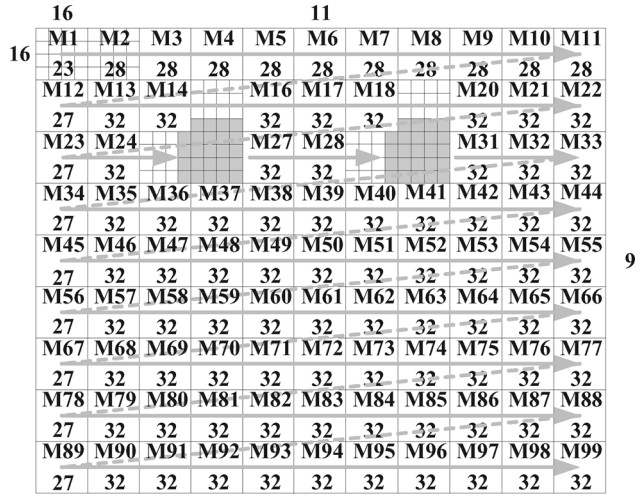

Fig. 12. Macroblock order and the number of block cycles for luma.

(cluster 2 to cluster 11) block cycles. The number of block processing cycle is shown for each macroblock in Fig. 14. As a result, the number of the total block cycles needed to process a QCIF frame is 2031 (one luma) + 1282 (two chroma) = 3313 (13 252 clock cycles). Table IV shows the data flow of the vertical processing order.

D. Configuration 4: Parallel Processing Architecture

In order to speed up the processing time (the fourth configuration), the EFO unit can be extended to a parallel engine. As shown in Fig. 15, both the horizontal and vertical filter units can be extended to four EFO units, and the memory bus can also be extended to 128-bit width. Therefore, the performance can achieve 4 times when compared to configuration 3. As a result, the number of the total cycles needed to process a QCIF frame is 3313 (Luma is 2031, two Chroma are $641 \times 2 = 1282$).

TABLE III
DATA FLOW OF SPA PROCESSING ORDER

State	5	6	7	8	9	10	11	12	13
A1	E5	B1	B2	B3	B4	E6	B5	B6	B7
T1	E4	E5	B1	B2	B3	B4	E6	B5	B6
W1	E3	E4	E5	B1	B2	B3	B4	E6	B5
W2	E2	E3	E4	E5	B1	B2	B3	B4	E6
W3	E1	E2	E3	E4	E5	B1	B2	B3	B4
W4		E1	E2	E3	E4	E5	B1	B2	B3
W5			E1	E2	E3	E4	E5	B1	B2
T2				E1	E2	E3	E4	E5	B1
MEM					E1	E2	E3	E4	E5
V		V6	V7	V8	V9		V11	V12	V13
H				H8	H9	H10	H11		H13

E. Configuration 5: Configurable Window-Based Architecture

The most cost-efficient approach is to use memory FIFO instead of register array A1, W1, W2, W3, W4, and W5, as shown in Fig. 16 (the fifth configuration), the configurable window-based architecture. The advantages of configurable window-based architecture are lower hardware cost, lower power consumption, and flexibility (configurable at any size of window).

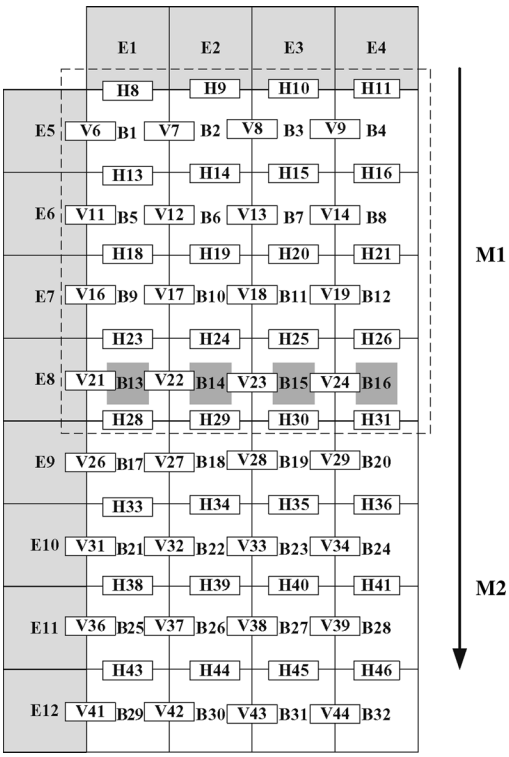

Fig. 13. Buffering scheme.

TABLE IV
DATA FLOW OF VERTICAL PROCESSING ORDER

State	Buf	Buf	Buf	Buf	1	2	3	4
Clock	21	22	23	24	25	26	27	28
A1	B13	B14	B15	B16	E9	B17	B18	B19
T1	E8	B13	B14	B15	B16	E9	B17	B18
W1	B12	E8	B13	B14	B15	B16	E9	B17
W2	B11	B12	E8	B13	B14	B15	B16	E9
W3	B10	B11	B12	E8	B13	B14	B15	B16
W4	B9	B10	B11	B12	E8	B13	B14	B15
W5	E7	B9	B10	B11	B12	E8	B13	B14
T2	B8	E7	B9	B10	B11	B12	E8	B13
MEM	B7	B8	E7	B9	B10	B11	B12	E8
V	V21	V22	V23	V24		V26	V27	V28
H	H21		H23	H24	H25	H26		H28

For luma blocks of a QCIF frame ($44 \times 36 = 1584$ basic 4×4 blocks) using the advanced window processing approach, the window size is configured to 44. As shown in Fig. 17, it requires 44 block cycles to load and process the first phase (the initial phase). At the 47th block cycle, block B1 completes all adjacent filtering (edge V2 and H47) and then it is written back to the internal memory in the next block cycle (the 48th block cycle). After that, B2 follows and so on. Table V shows the data flow of advanced 44 windows processing. There are five states and two filtering operations, as shown in Table V. For the first state A1 (at the 47th block cycle), the basic 4×4 block data (B47) is fetched from the internal memory to A1 FIFO and is filtered with adjacent block B46 for vertical edge V47 using a horizontal filtering unit. At second state T1 (at the 48th block cycle), block B48 is fetched from the internal memory and filtered with block B47, and then transposes the filtered block B47 from vertical to horizontal and store the transposed block B47 to T1 register array. At the window state W1 (at the 49th block cycle), block

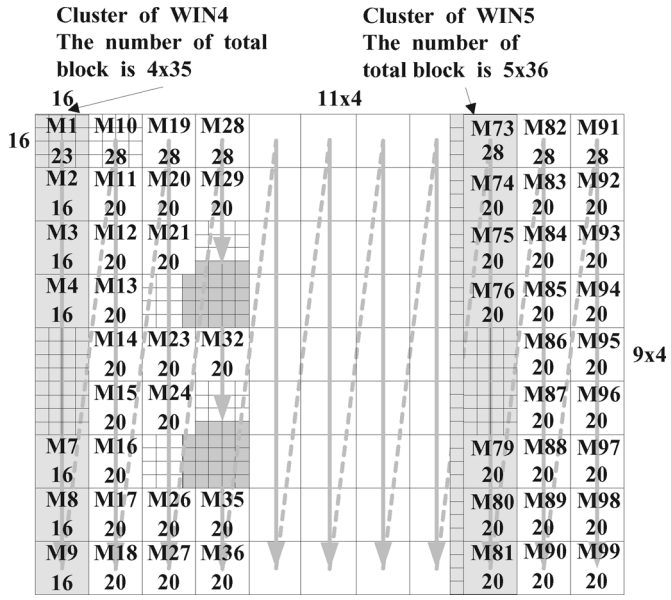

Fig. 14. Vertical processing order and the number of block cycles for luma.

B47 is filtered with block B3 for the edge H49. Now, block B3 has completed all filtering with adjacent block B2, B4, and B47 (left, right, and button). At the same time, the horizontal filtering unit also performs the filtering for vertical edge V49 between block B48 and block B49. Then the system writes block B3 to the internal memory at the 50th block cycle. Then edge V50, H50, and the rest follow in the same way. As a result, the number of total processing time for all luma blocks using the WIN44 architecture is $47 + 44 \times 36 = 1631$ block cycles (6524 cycles)

For chrominance, there are two QCIF chroma blocks ($(22 \times 18) \times 2$ basic 4×4 blocks) to be processed. The processing approach is the same as the luma blocks but the window size is configured to 22 (WIN22). The processing order and data flow are presented in Fig. 18 and Table VI respectively. The initial processing time for each chroma block using the WIN22 is 25 block cycles. At the 26th block cycle, the first block B1 has finished two times of filtering and will be written to the internal memory at the 27th block cycle. After that, B2 follows and so on. As a result, the number of total processing time for two chroma blocks using the WIN22 architecture is $(25 + 22 \times 18) \times 2 = 842$ block cycles (3368 cycles). Therefore, the total number of processing time for a QCIF (one luma and two chroma) using WIN44 and WIN22 is $1631 + 842 = 2473$ block cycles (9892 cycles).

According to different image resolution, the configurable window-based architecture can be configured to any size of windows by setting the window registers, as shown in Table VII. Moreover, the performance can achieve a $4\times$ speed improvement, when both the horizontal and vertical filter unit are extended to 4 EFO units (WIN44Parallel).

V. RESULT

The simulation results are shown in Table VIII. The architecture of WIN as a coprocessor can accelerate the H.264/AVC decoder system. Moreover, the number of total memory references for load and store is reduced by 34% and 36%, respectively.

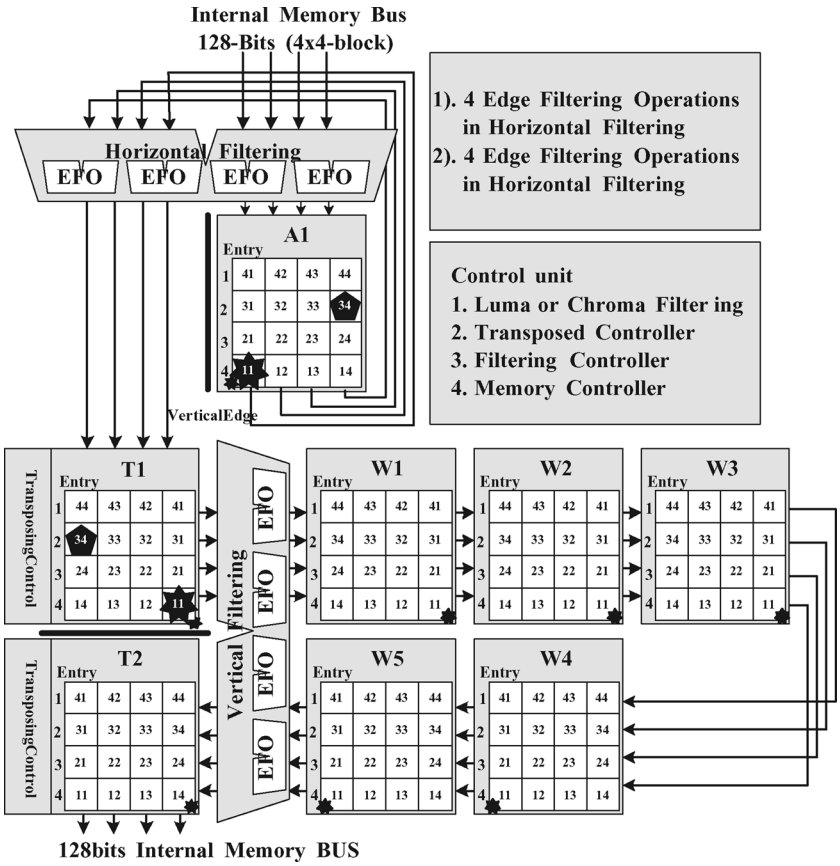

Fig. 15. Parallel processing configuration.

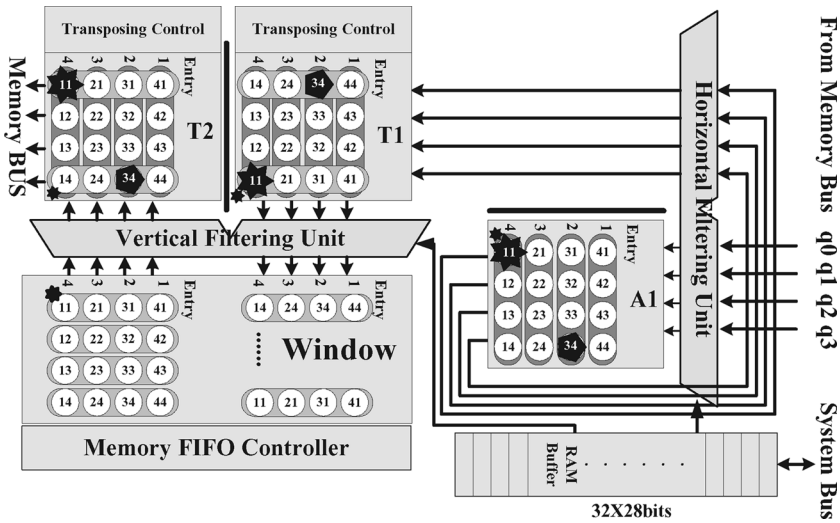

Fig. 16. Configurable window-based architecture.

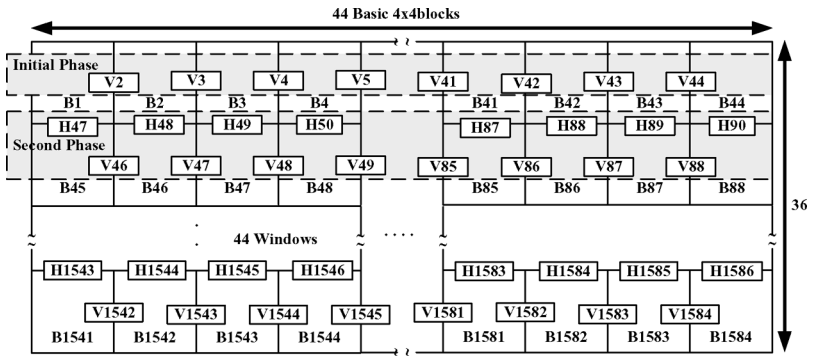

Fig. 17. Advanced window processing order for WIN44.

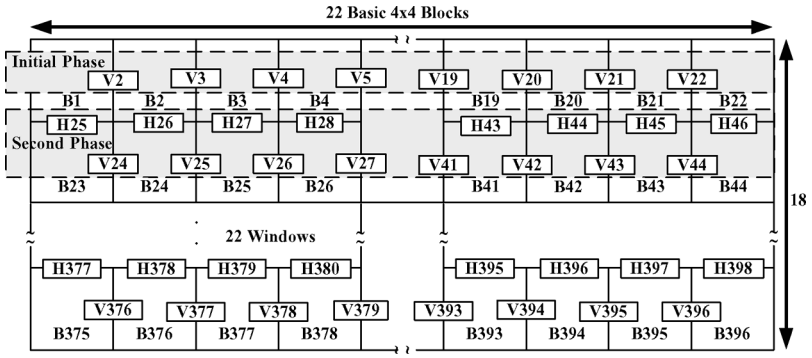

Fig. 18. Advanced window processing order for WIN22.

TABLE V
DATA FLOW OF ADVANCED 44 WINDOWS PROCESSING

State	47	48	49	50	51	52	53	54
A1	B47	B48	B49	B50	B51	B52	B53	B54
T1	B46	B47	B48	B49	B50	B51	B52	B53
W1	B45	B46	B47	B48	B49	B50	B51	B52
W2	B44	B45	B46	B47	B48	B49	B50	B51
..	..	..	..	..	..	..	..	..
W43	B3	B4	B5	B6	B7	B8	B9	B10
W44	B2	B3	B4	B5	B6	B7	B8	B9
T2	B1	B2	B3	B4	B5	B6	B7	B8
MEM		B1	B2	B3	B4	B5	B6	B7
V	V47	V48	V49	V50	V51	V52	V53	V54
H	H47	H48	H49	H50	H51	H52	H53	H54

TABLE VI
DATA FLOW OF ADVANCED 22 WINDOWS PROCESSING

State	25	26	27	28	29	30	31	32
A1	B25	B26	B27	B28	B29	B30	B31	B32
T1	B24	B25	B26	B27	B28	B29	B30	B31
W1	B23	B24	B25	B26	B27	B28	B29	B30
W2	B22	B23	B24	B25	B26	B27	B28	B29
..	..	..	..	..	..	..	..	..
W21	B3	B4	B5	B6	B7	B8	B9	B10
W22	B2	B3	B4	B5	B6	B7	B8	B9
T2	B1	B2	B3	B4	B5	B6	B7	B8
MEM		B1	B2	B3	B4	B5	B6	B7
V	V25	V26	V27	V28	V29	V30	V31	V32
H	H25	H26	H27	H28	H29	H30	H31	H32

TABLE VII
NUMBER OF BLOCK CYCLES FOR VARIOUS RESOLUTIONS

Resolution	Win Size	Luma	2Chroma	Total
QCIF 176x144	44&22	1631	842	2473
CIF 352x288	88&44	6427	3262	9689
VGA 640x480	160&80	19363	9766	29129
Video 1280x720	320&160	58923	29126	88049
HDTV 1920x1080	480&240	130083	65286	195369

TABLE VIII
PERFORMANCE COMPARISON

Item	Software	WIN	Reduced by
Inst.	128640967	75123050	42%
Load	30443106	20180448	34%
Store	16098837	10295823	36%
Branch	14324486	7901023	49%
Cycles	220929397	132532824	40%

A. Memory Performance

Using our proposed window-based architecture, the memory performance is improved by four times when compared to the software implementation. Table IX shows the comparison of various architectures to process a macroblock (one 16×16 luma block and two 8×8 chroma blocks). The memory access times of our WIN architecture using novel processing are reduced by 600 to 64, when compared to the previous design in [24], [28], [30], and [34]. In other words, the reduced memory reference will result in higher performance and better power consumption.

TABLE IX
MEMORY REFERENCE OF VARIOUS ARCHITECTURES

Author	Architecture	MEM
JM9.2[36]	Software Implementation	(65+17+17)x4x2=792
Huang[24]	Basic+Single-port SRAM	792
Huang[24]	Basic+Two-port SRAM	792
Huang[24]	Advance+Dual-port SRAM	(40+12+12)x4x2=512
Huang[24]	Dual Arrays+Two-port SRAM	512
Li[34]	5120 bits Dual-port SRAM	(96x2)x2=384
Chen[28]	Dual-port SRAM	(24+8+8)x4x2=320
Chen[30]	Dual-port SRAM	(20+6+6)x4x2=256
WIN44	Dual-port SRAM	(16+4+4)x4x2=192

B. System Performance

As shown in the previous section, when using 44 and 22 window size and parallel architecture for luma and chroma respectively, the filtering for a QCIF frame takes 1631 and $421 \times 2 = 842$ cycles respectively. As a result, the total filtering takes 2473 cycles for a QCIF frame. Our filtering scheme takes fewer cycles when compared to $294 \times 99 = 29\,016$, $286 \times 99 = 28\,314$, $240 \times 99 = 23\,760$, and $192 \times 99 = 19\,008$ cycles of the architecture described in [24], [26], [28], and [34]. Table X shows the performance comparison of various architectures.

C. Implementation

We implemented the configurable window-based architecture by VHDL language and synthesized the design using TSMC 0.18 μm Artisan CMOS cell library with Synopsys Design Compiler by setting the critical path constraint to 10 ns (100 MHz). The hardware comparison of the various architectures is shown in Table XI.

D. Verification

In order to verify our configurable window-based architecture, we modified JM9.2 to fit our test platform and implemented our proposed architecture on FPGA, as shown in Fig. 19. The data file of the reconstructed pixel before filtering is saved in

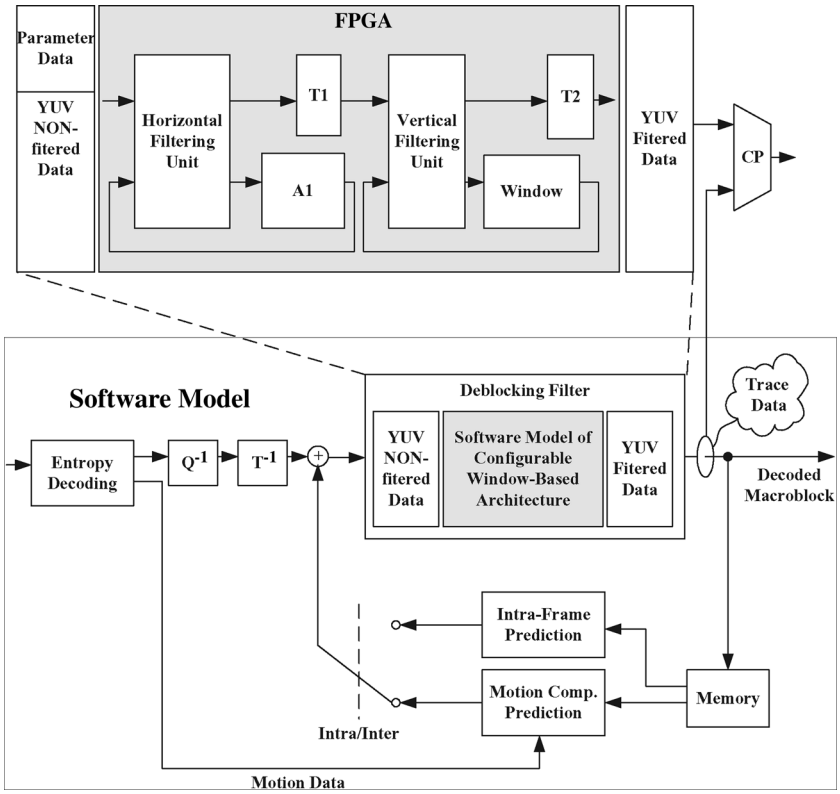

Fig. 19. Block diagram of verification.

TABLE X
PERFORMANCE COMPARISON OF VARIOUS ARCHITECTURES

Author	Architecture	Cycles
Huang[24]	Basic+Single-port SRAM	504x99=49896
Huang[24]	Advance+Dual-port SRAM	440x99=43560
Huang[24]	Basic+Two-port SRAM	408x99=40392
Huang[24]	Dual Arrays+Two-port SRAM	294x99=29016
Sheng[26]	2-D Deblocking Filter	286x99=28314
Chen[28]	Dual-port SRAM	240x99=23760
Li [34]	5120 bits Dual-Port SRAM	192x99=19008
CFG1-EFO	EFO with 64-bits data bus	192x99=19008
CFG2-SPA	SPA with 2 EFO (32-bits data)	5537x4=22148
CFG3-SPA	SPA with vertical processing	3313x4=13252
CFG4-Parallel	SPA with 8 EFO (128-bits data)	3313
CFG5-WIN44	Dual-port SRAM(32-bits data)	9892
WIN44Parallel	Dual-port SRAM(128-bits data)	2473

TABLE XI
HARDWARE COMPARISON OF VARIOUS ARCHITECTURES

Author	Architecture	Gate Count
Huang[24]	Basic+Single-port SRAM	18.91K
Huang[24]	Advance+Dual-port SRAM	20.66K
Li [34]	5120 bits Dual-Port SRAM	9.57K
Chen[27]	Dual-port SRAM	5.66K
Chen[28]	Dual-port SRAM	22K
CFG1-EFO	EFO with computation of Bs	6.3K
CFG2-SPA	SPA with 2 EFO (32-bits data bus)	20.84K
CFG3-SPA	SPA with vertical processing(32-bits)	20.84K
CFG4-Paral	SPA with 8 EFO (128-bits data)	58.64K
CFG5-WIN44	Dual-port SRAM(32-bits data)	14.75K
WIN44Paral	Dual-port SRAM(128-bits data)	52.55K

YUV nonfiltered data RAM, which is driven from the software model and then sent to the deblocking filter architecture under control. The YUV filtered results are compared with the filtered results of software module.

VI. CONCLUSION

In this paper, we propose a configurable window-based VLSI architecture to accelerate the operations of deblocking filter for H.264/AVC video coding. The major idea is to reduce the number of memory references through the simultaneous processing architecture WIN using a novel processing order. As a result, the processing capability of the proposed architecture is very appropriate for real-time deblocking of high-definition television (HDTV, 1920 $\times$ 1080 pixels/frame, 60 frames/s video signals) video operating at 60 MHz, and the area is very small (around 14.75 K logic gates + $481 \times 4 \times 32$ – bit memory FIFO).

REFERENCES

[1] *Advanced Video Coding for Generic Audiovisual Services*, ITU-T Recommendation H.264, Mar. 2003, , .
[2] ITU-T Recommendation H.261, Video Codec for Audiovisual Services at p X 64 kbit/s Mar. 1993.
[3] *Information Technology-Coding of Moving Pictures and Associated Audio for Digital Storage Media at Up to About 1.5 Mbit/s*, ISO/IEC 11172, 1993, Geneva.
[4] *Generic Coding of Moving Pictures and Associated Audio Information-Part 2: Video Also ITU-T Recommendation H.262*, ISO/IEC 13818-2, 1994.
[5] *Video Coding for Low Bit Rate Communication*, ITU-T Recommendation H.263, 1998.
[6] *Information Technology-Coding of Audiovisual Objects-Part 2: Visual*, ISO/IEC 14496-2, 2000.
[7] P. List, A. Joch, J. Lainema, G. Bjntegaard, and M. Karczewicz, "Adaptive deblocking filter," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 614–619, Jul. 2003.
[8] H. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, "Low-complexity transform and quantization in H.264/AVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 598–603, Jul. 2003.
[9] N. Ahmed, T. Natarajan, and R. Rao, "Discrete cosine transform," *IEEE Trans. Comput.*, vol. C-23, no. 1, pp. 90–93, Jan. 1974.

[10] T. Wiegand, X. Zhang, and B. Girod, "Long-term memory motion-compensated prediction for video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, no. 2, pp. 70–84, Feb. 1999.
[11] M. Karczewicz and R. Kurceren, "The SP and SI frames design for H.264/AVC," *IEEE Trans. Circuits Syst.*, vol. 13, no. 7, pp. 637–644, Jul. 2003.
[12] T. Wiegand, H. Schwarz, A. Joch, and F. Kossentini, "Rate-constrained coder control and comparison of video coding standards," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 688–703, Jul. 2003.
[13] T. Wedi and H. G. Musmann, "Motion- and aliasing-compensated prediction for hybrid video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 577–587, Jul. 2003.
[14] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 620–636, Jul. 2003.
[15] J. Ribas-Corbera, P. A. Chou, and S. Regunathan, "A generalized hypothetical reference decoder for H.264/AVC," *IEEE Trans. Circuits Syst.*, vol. 13, no. 7, pp. 674–687, Jul. 2003.
[16] B. Girod, M. Kalman, Y. J. Liang, and R. Zhang, "Advances in video channel-adaptive streaming," presented at the ICIP 2002, Rochester, NY.
[17] Y. J. Liang and B. Girod, "Rate-distortion optimized low-latency video streaming using channel-adaptive bitstream assembly," presented at the ICME 2002, Lausanne, Switzerland.
[18] S. H. Kang and A. Zakhor, "Packet scheduling algorithm for wireless video streaming," presented at the Int. Packet Video Workshop, Pittsburgh, PA, Apr. 2002.
[19] S. Wenger, "H.264/AVC over IP," *IEEE Trans. Circuits Syst.*, vol. 13, no. 7, pp. 645–656, Jul. 2003.
[20] T. Stockhammer, M. M. Hannuksela, and T. Wiegand, "H.264/AVC in wireless environments," *IEEE Trans. Circuits Syst.*, vol. 13, no. 7, pp. 657–673, Jul. 2003.
[21] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi, "Video coding with H.264/AVC: Tools, performance, and complexity," *IEEE Circuits Syst. Mag.*, vol. 4, no. 1, pp. 7–28, First Quarter, 2004.
[22] M. I. T. Orchard and G. J. Sullivan, "Overlapped bock motion compensation: An estimation-theoretic approach," *IEEE Trans. Image Process.*, vol. 3, no. 5, pp. 693–699, Sep. 1994.
[23] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro, "H.264/AVC baseline profile decoder complexity analysis," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 715–727, Jul. 2003.
[24] Y. W. Huang, T. W. Chen, B. Y. Hsieh, T. C. Wang, T. H. Chang, and L. G. Chen, "Architecture design for de-blocking filter in H.264/JVT/AVC," in *Proc. IEEE Conf. Multimedia Expo*, 2003, pp. 693–696.
[25] M. Sima, Y. Zhou, and W. Zhang, "An efficient architecture for adaptive deblock filter of H.264/AVC video coding," *IEEE Trans. Consum. Electron.*, vol. 50, no. 1, pp. 292–296, Feb. 2004.
[26] B. Sheng, W. Gao, and D. Wu, "An implemented architecture of deblocking filter for H.264/AVC," in *Proc. IEEE Int. Conf. Image Process. (ICIP'04)*, Oct. 2004, vol. 1, pp. 665–668.
[27] C. M. Chen and C. H. Chen, "An Efficient VLSI architecture of edge filtering in H.264/AVC," in *Proc. IASTED Int. Conf. Circuits, Signals, Syst.*, Oct. 2005, pp. 118–122.
[28] C. M. Chen and C. H. Chen, "An efficent architecture for deblocking filter in H.264/AVC video coding," in *Proc. IASTED Int. Conf. Comput. Graphics Imaging*, Aug. 2005, pp. 177–181.
[29] C. M. Chen and C. H. Chen, "Parallel processing for deblocking filter in H.264/AVC," in *Proc. IASTED Int. Conf. Commun., Internet, Inf. Technol.*, Oct. 2005, pp. 188–191.
[30] C. M. Chen and C. H. Chen, "A memory efficient VLSI architecture for deblocking filter in H.264 using vertical processing order," in *Proc. IEEE Int. Conf. Intell. Sensors, Sensor Netw. Inf. Process.*, Dec. 2005, pp. 361–366.
[31] C. M. Chen and C. H. Chen, "Window architecture for deblocking filter in H.264/AVC," in *Proc. IEEE Int. Symp. Signal Process. Inf. Technol.*, Vancouver, BC, Canada, Aug. 2006, pp. 338–342.
[32] C. M. Chen, C. H. Chen, J. P. Zeng, W. C. Hsu, and C. T. Yu, "Windows processing for deblocking filter in H.264/AVC," in *Proc. 32th Annu. Conf. IEEE Ind. Electron. Soc.*, Paris, France, Nov. 2006, pp. 3428–3433.
[33] C. M. Chen and C. H. Chen, "An efficient pipeline architecture for deblocking filter in H.264/AVC," *IEICE Trans. Inf. Syst.*, vol. E90-D, no. 1, pp. 99–107, Jan. 2007.
[34] L. Li, S. Goto, and T. Ikenaga, "A highly parallel architecture for deblocking filter in H.264/AVC," *IEICE Trans. Inf. Syst.*, vol. E88-D, no. 7, pp. 1623–1629, Jul. 2005.
[35] D. C. Burger and T. M. Austin, SimpleScalar Tool Set, Version 2.0 Univ. Wisconsin, Madison, 1997, Tech. Rep..
[36] H.264/AVC Reference Software JM9.2.

Chung-Ming Chen received the M.S. degree in electronic engineering from National Yunlin University of Science and Technology, Taiwan, R.O.C., in 1996 and the Ph.D. degree from Department of Electrical Engineering, National Cheng Kung University, Taiwan, in 2007.

He joined Elan Corp. (Fabless Corp.), Hsinchu, Taiwan, in 2000, where he is currently Project Leader for the System Chip Design Division. His research interests include computer architecture, VLSI architecture of video-coding, DSP architecture design, and network processors.

Chung-Ho Chen received the M.S.E.E. degree in electrical engineering from the University of Missouri, Rolla, in 1989 and the Ph. D. degree in electrical engineering from the University of Washington, Seattle, in 1993.

He was a Faculty Member of the Department of Electronic Engineering, National Yunlin University of Science and Technology. In 1999, he joined the Department of Electrical Engineering, National Cheng Kung University, Tainan City, Taiwan, R.O.C., where he is currently a Professor. His research areas include advanced computer architecture, video technology, and network storages. He coholds a U.S. patent on a multicomputer cluster-based processing system and a R.O.C. patent on a multiple-protocol storage structure.

Dr. Chen was the Technical Program Chair of the 2002 VLSI Design/CAD Symposium held in Taiwan.