PAPER *Special Section on Advanced Image Technology*

An Efficient Pipeline Architecture for Deblocking Filter in H.264/AVC

Chung-Ming CHEN[†a)], ***Member and*** **Chung-Ho CHEN**[†b)], ***Nonmember***

SUMMARY In this paper, we study and analyze the computational complexity of 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 novel processing order with efficient VLSI 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 the software implementation. Moreover, the system performance of our design significantly outperforms the previous proposals.

key words: *deblocking filter, H.264/AVC, video coding*

1. Introduction

Video compression is a critical component in today's multimedia systems. The limited transmission bandwidth or storage capacity for applications such as DVD, digital television, or 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 (H.261 [2], MPEG-1 Video [3], MPEG-2 Video [4], H.263 [5], and MPEG-4 Visual or part2 [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 like 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 advanced functionality 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.

Manuscript received April 10, 2006.
Manuscript revised June 15, 2006.
[†]The authors are with the Department of Electrical Engineering, National Cheng Kung University, Taiwan.
a) E-mail: cmchen@ee.ncku.edu.tw
b) E-mail: chchen@mail.ncku.edu.tw
DOI: 10.1093/ietisy/e90–d.1.99

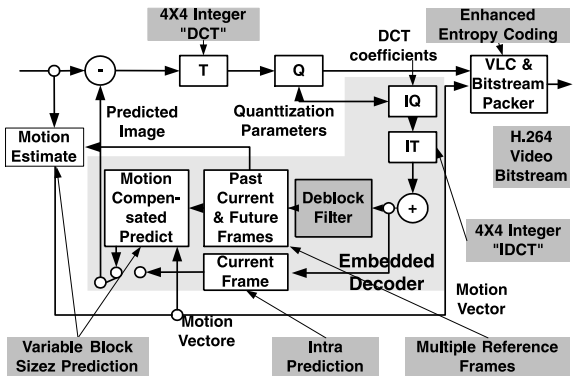

Fig. 1 Block diagram of H.264/AVC.

As our experiment result indicates, the operation of the deblocking filter is the most time consuming part of the 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 5–10% while producing the same objective quality as the non-filtered video [23]. Adaptive deblocking filter can also be used in inter-picture 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 the intensive computations, in [24]–[30] and [31] 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 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 computa-

tion of boundary strength operations into edge filtering 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 the software implementation [1]. The proposed architecture is called Simultaneous Processing Architecture (SPA). It uses a novel processing order within a macroblock 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 Sect. 2, the algorithm of the deblocking filter is explained. Section 3 analyzes the computational complexity of H.264/AVC baseline decoder. Section 4 illustrates the block diagram of our proposed architecture using a novel processing order. Section 5 shows the simulation results. Finally, conclusion is presented in Sect. 6.

2. Algorithm of Deblocking Filter

In this section, we briefly describe the algorithm of the 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].

2.1 Processing Order

As in the H.264/AVC standard recommendation [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 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. 2.

According to this rule, there are three types of processing orders which are proposed by [24] and [26] as shown in Figs. 3, 4, and 5. It is obvious that an adaptive deblocking filter should 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 an efficient VLSI architecture with a novel processing order to reduce the memory reference of each 4×4-block to one as shown in Sect. 4.

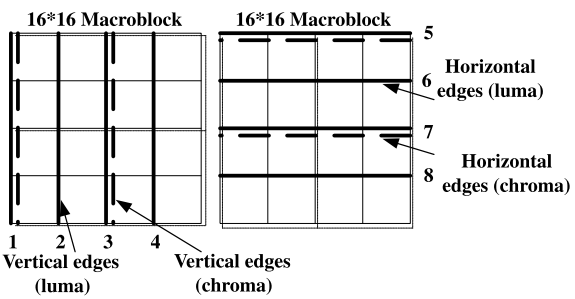

Fig. 2 Processing method of standard.

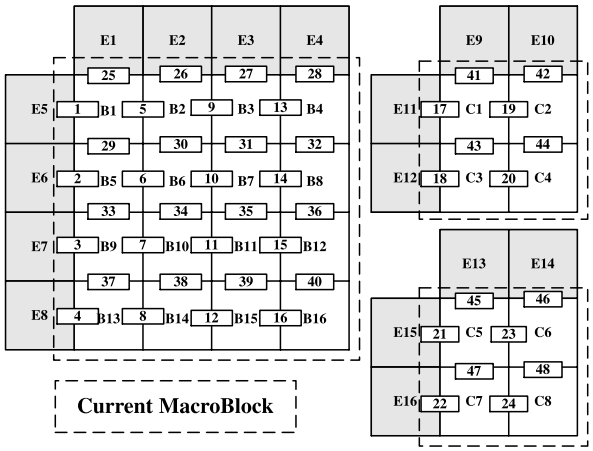

Fig. 3 Basic processing order of [24].

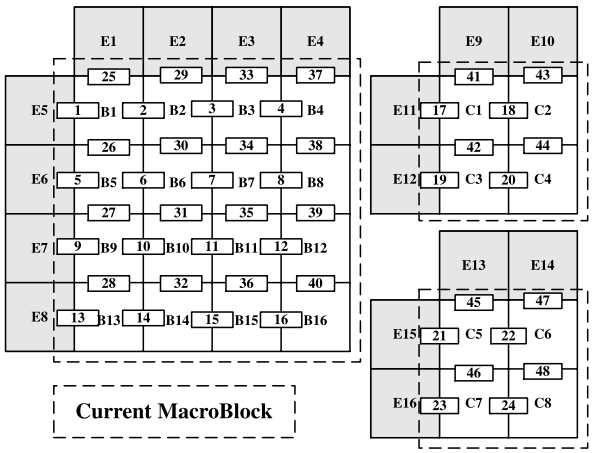

Fig. 4 Advanced processing order of [24].

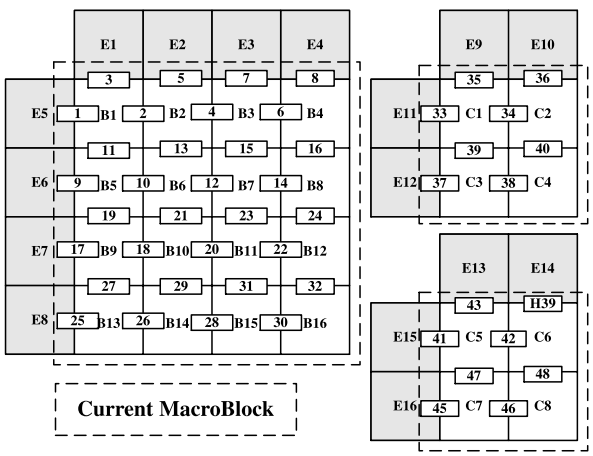

Fig. 5 Processing order of [26].

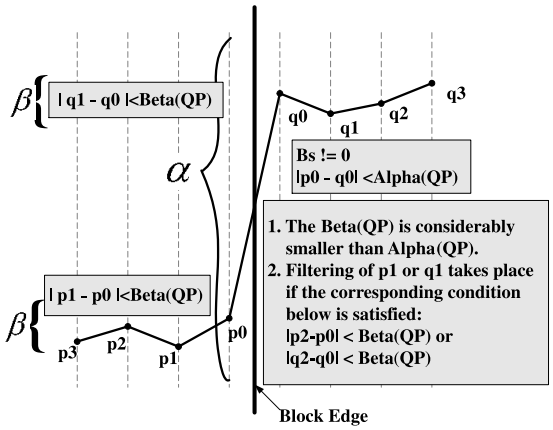

Fig. 6 Principle of deblocking filter.

2.2 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. 6 illustrates the principle of the deblocking filter using a one-dimensional visualization of 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:

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

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

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

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

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

The dependency of Alpha and Beta on the quantizer, links the strength of filtering to the general quality of the reconstructed picture prior to filtering. For small quantizer values, the thresholds both become zero, and filtering is effectively turned off altogether.

3. Computational Complexity

The simulator used in this study is derived from the SimpleScalar/ARM tool set [32], 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 described in Table 1.

The H.264/AVC JM9.2 code [33] is used for reporting the complexity assessment of the experiments. The test sequences used in the computational complexity assessment are Forman 30 Hz QCIF and CIF, Mother and Daughter 30 Hz QCIF, Mobile and Akiyo 30 Hz CIF. A fixed quantization parameter setting with a QP of 28 has been assumed.

Table 1 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 16 KB
I-Cache	32-way, LRU, 1-cycle hit, total 8 KB
Memory Latency	12
Memory Width	4 bytes

Table 2 Computational complexity of decoder.

Function	Complexity
Deblocking Filtering	36%
Interpolation	22%
Entropy Coding	13%
Inverse Transfers and Reconstruction	13%
Others	14%

Table 3 The complexity of the deblocking filter.

Function	Complexity
Computation of Boundary Strength	40%
Filtering Operations	60%

One of the most important issues in computational complexity of H.264/AVC decoder is the distribution of time complexity among its major sub-functions. In our simulation result, as shown in Table 2, deblocking filtering (36%) is the largest component, followed by interpolation (22%), bitstream parsing and entropy decoding (13%), and inverse transfers and reconstruction (13%). The computational complexity of the deblocking filter is shown in Table 3.

3.1 Computation of Boundary Strength

As our experimental result indicates, the operation of the deblocking filter, which is the most time consuming part of an H.264/AVC decoder, can be separated into two major sub-functions. The first sub-function is the computation of the Boundary Strength (Bs) parameter for each edge filtering operation. The purpose of this computation is to determine whether a block artifact may have been produced across the boundary, and thus determine the strength (Bs) of the filter to be used on the edge. A Boundary Strength (Bs) is assigned an integer value from 0 to 4. A strongest filter (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.

3.2 Edge Filtering Operation

The second important sub-function is the content activity check and filtering operations as shown in the Eqs. (1) to (4) and the Eqs. (5) to (19) respectively. In order to separate the true edge and blocking artifact, the sample values across every edge to be filtered are analyzed. As stated in Sect. 2, filtering does not take place for edges with Bs equal to zero. For edges with nonzero Bs values, a pair of quantization-dependent parameters, referred to as Alpha and Beta, are used in the content activity check that determines whether each set of samples is filtered. Both table-derived threshold Alpha and Beta are dependent on the average quantization parameter (QP) employed over the edge, as well as encoder selected offset values that can be used to control the properties of the deblocking filter on the slice level. The filtering operations are shown in the following:

Luma4 for Q samples: (Bs=4)

$$q'_0 = (p_1 + 2p_0 + 2q_0 + 2q_1 + q_2 + 4) >> 3 \quad (5)$$

$$q'_1 = (p_0 + q_0 + q_1 + q_2 + 2) >> 2 \quad (6)$$

$$q'_2 = (2q_3 + 3q_2 + q_1 + q_0 + p_0 + 4) >> 3 \quad (7)$$

Luma4 for P samples: (Bs=4)

$$p'_0 = (q_1 + 2q_0 + 2p_0 + 2p_1 + p_2 + 4) >> 3 \quad (8)$$

$$p'_1 = (q_0 + p_0 + p_1 + p_2 + 2) >> 2 \quad (9)$$

$$p'_2 = (2p_3 + 3p_2 + p_1 + p_0 + q_0 + 4) >> 3 \quad (10)$$

Chroma4 for P and Q samples: (Bs=4)

$$q'_0 = (2q_1 + q_0 + p_1 + 2) >> 2 \quad (11)$$

$$p'_0 = (2p_1 + p_0 + q_1 + 2) >> 2 \quad (12)$$

Luma and Chroma: (Bs=3, 2, and 1)

$$Dif = Clip(-t_c, t_c, ((((q_0 - p_0) << 2) + (p_1 - q_1) + 4) >> 3)) \quad (13)$$

$$p'_0 = Clip(p_0 + Dif) \quad (14)$$

$$q'_0 = Clip(q_0 - Dif) \quad (15)$$

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

$$Dif = Clip(-t_{c0}, t_{c0}, (p_2 + ((p_0 + q_0 + 1) >> 1) - (p_1 << 1)) >> 1) \quad (16)$$

$$p'_1 = p_1 + Dif \quad (17)$$

$$Dif = Clip(-t_{c0}, t_{c0}, (q_2 + ((p_0 + q_0 + 1) >> 1) - (q_1 << 1)) >> 1) \quad (18)$$

$$q'_1 = q_1 + Dif \quad (19)$$

4. Proposed Architecture

The key features of our proposed architecture can be divided into two major components, including the edge filtering unit and a simultaneous processing engine that employs a novel processing order to simultaneously process the horizontal filtering of vertical edge and vertical filtering of horizontal edge.

4.1 Edge Filtering Unit

The complexity of the H.264/AVC deblocking filter is mainly based on two reasons. The first reason is the highly adaptive filtering, which requires several conditional processing on each block edges and sample levels. As described in the previous section, the computation of boundary strength, the threshold value of Alpha and Beta, the table-derived operations, and edge filtering operation are known to be very time consuming. Therefore, we propose an efficient VLSI architecture that includes content activity check operations, the table-derived operations, filtering operations, and computation of boundary strength operations into the edge filter unit to accelerate the horizontal and vertical filtering on the boundary of two adjacent basic 4×4 blocks as shown in Fig. 7. A detailed description of the edge filtering unit can be found in [27]. There are five major sub-functions in our Edge Filtering Unit architecture as described below.

- **The Computation of Boundary Strength:** The purpose of this computation 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 detailed description of the computation of boundary strength can be found in Sect. 3. For each boundary between neighboring 4×4-luma blocks, a boundary strength assign-

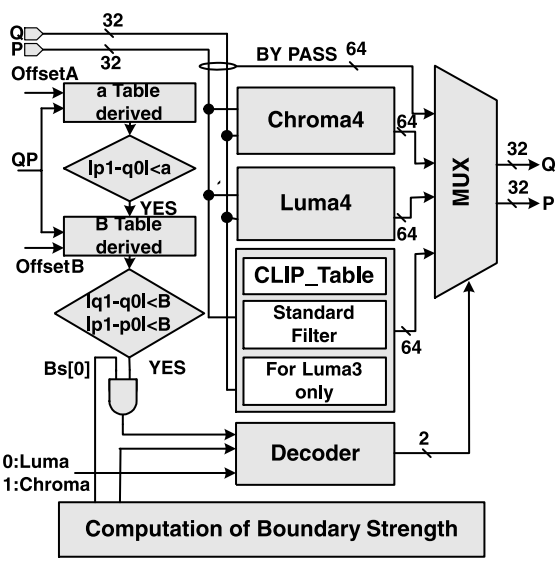

Fig. 7 Edge filter unit.

ment is implemented as shown in Fig. 8.

- **The Filtering Operation:** The most important function of deblocking filter is the filtering operation, which is divided into two modes. A special mode of filtering that allows for stronger filtering is applied when Bs is equal to 4 as shown in the Eqs. (5) to (12). The others are standard mode of filtering with a Bs parameter of 1 to 3 as shown in the Eqs. (13) to (19).
- **Clipping Operation:** The filtering operation would 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. For Eqs. (14) and (15), the result value of P and Q are limited between 0 and 255. The value of t_{c0} is derived from the Clip Table. The threshold t_c is determined as follows:

  If the edge is Luma

$$t_c = t_{c0} + ((\alpha_p < \beta)?1 : 0) + ((\alpha_q < \beta)?1 : 0) \quad (20)$$

  Otherwise the edge is chroma

$$t_c = t_{c0} + 1 \quad (21)$$

- **Content Activity Check Operation:** Conditional branches which are described in the Eqs. (1) to (4) almost inevitably appear in the inner most loops of the algorithm. So, we implement it into our Edge Filtering Unit.
- **Table-derived Operations:** In order to simultaneously access Alpha, Beta, and Clip tables, since most values of these tables are zero, we used combinational logic to implement Alpha, Beta, and Clip tables instead of using memory buffer. This can save most of the space of memory buffer and improve overall system performance and then reduce power consumption.

4.2 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 2 samples in each direction, each sample in a picture must be transferred from and to internal memory 4 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 an efficient architecture, which can simultaneously processes the horizontal filtering of vertical edge and vertical filtering of horizontal edge as shown in Fig. 9. The proposed architecture is called Simultaneous Processing Architecture (SPA).

There are three major sub-functions in our proposed architecture. The first component is the Shift Operation Array. There are six forwarding shift register arrays in our proposed architecture (for example, Array1, 3, 4, 5, 6, and 7). Each array contains four entries, each with 4 processed samples. And the shift direction is shown in Fig. 9. The second sub-function is the transposing operation as shown in Fig. 9. The Array2 and Array8 latch the 4×4 block sample values that are transposed from Array1 and Array7 respectively. The final important functions are the horizontal and vertical filter units which are described in the previous subsection.

4.3 Novel Processing Order in Macroblock

In a 16×16 sample macroblock, our architecture utilizes a novel processing order, which allows the simultaneous processing of horizontal and vertical filtering as shown in Fig. 10. The processing order begins from V6 to V7 (V denotes Vertical edge and 6 represents the sixth block cycle). And then at the eighth block cycle, the vertical edge V8 and horizontal edge H8 (H denotes Horizontal edge and 8 repre-

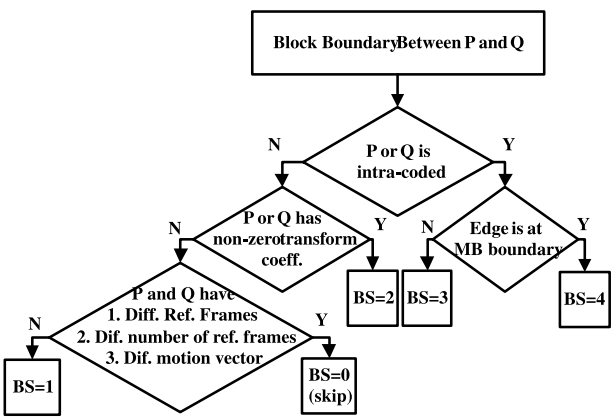

Fig. 8 Flowchart for determining boundary strength.

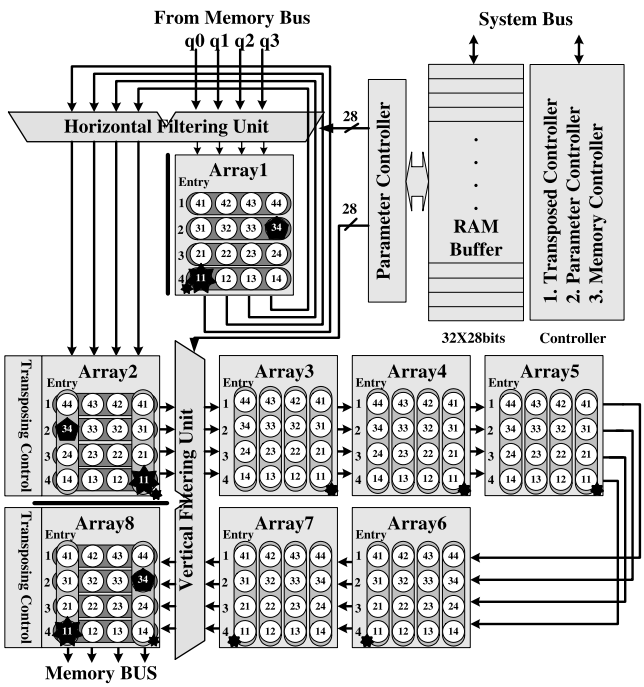

Fig. 9 Proposed architecture.

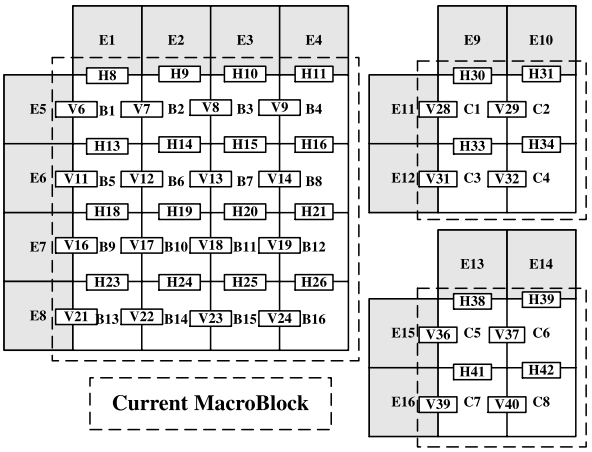

Fig. 10 Proposed processing order.

sents the eighth block cycle) are simultaneously processed with the horizontal and vertical filtering unit. Then V9, H9 follows, so on and so forth.

4.4 Basic Processing

In this subsection, the basic processing using raster scan order for a picture is shown in Fig. 11. The data flow for the raster scan processing order to process each luma macroblock is presented in Table 4. For a basic 4×4-block, it takes 13 block cycles (52 clock cycles) to process the first block B1. In the first 5 block cycles (the first 20 clock cycles), the blocks of E1, E2, E3, E4, and E5 are loaded from the internal memory to SPA's Array5, Array4, Array3, Array2, and Array1 respectively and no filtering operations are performed. In the next 2 block cycles (the sixth and seventh block cycle), 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 the 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, the 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 the block B1 to the internal memory at the fourteenth block cycle. The following block B2 will be completed at next block cycle. Then B3 follows, so on and so forth. Therefore, the number of total processing time for each luma macroblock is 32 block cycles (128 clock cycles).

4.5 Pipeline Processing

When processing the next macroblock M2, the advanced processing uses the pipeline to pre-load the initial blocks E9, E10, E11, E12 and B4 in SPA's Array5, Array4, Array3, Array2, and Array1 respectively as shown in Fig. 12. And

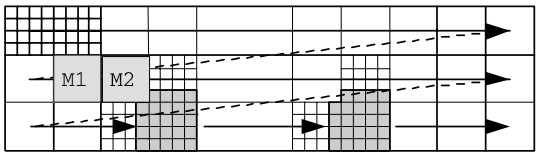

Fig. 11 Raster scan processing order.

Table 4 Data flow of raster scan processing order.

State	5	6	7	8	9	10	11	12	13
Array1	E5	B1	B2	B3	B4	E6	B5	B6	B7
Array2	E4	E5	B1	B2	B3	B4	E6	B5	B6
Array3	E3	E4	E5	B1	B2	B3	B4	E6	B5
Array4	E2	E3	E4	E5	B1	B2	B3	B4	E6
Array5	E1	E2	E3	E4	E5	B1	B2	B3	B4
Array6		E1	E2	E3	E4	E5	B1	B2	B3
Array7			E1	E2	E3	E4	E5	B1	B2
Array8				E1	E2	E3	E4	E5	B1
MEM					E1	E2	E3	E4	E5

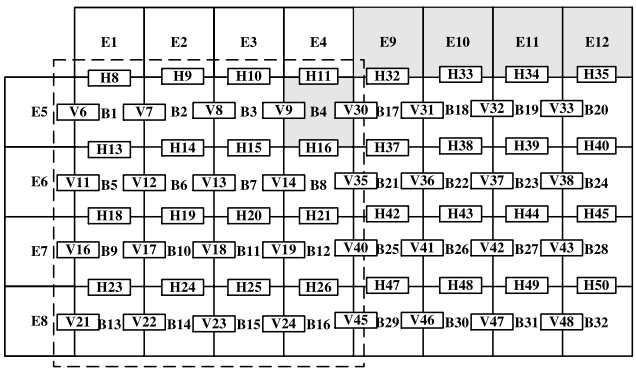

Fig. 12 Pre-load the initial blocks.

Table 5 Data flow of the pipeline processing method.

State	29	30	31	32	33	34	35
Array1	B4	B17	B18	B19	B20	B8	B21
Array2	E12	B4	B17	B18	B19	B20	B8
Array3	E11	E12	B4	B17	B18	B19	B20
Array4	E10	E11	E12	B4	B17	B18	B19
Array5	E9	E10	E11	E12	B4	B17	B18
Array6	B16	E9	E10	E11	E12	B4	B17
Array7	B15	B16	E9	E10	E11	E12	B4
Array8	B14	B15	B16	E9	E10	E11	E12
MEM	B13	B14	B15	B16	E9	E10	E11

thus we can save 5 block cycles when processing each macroblock. As a result, the number of the total block cycles to process a luma macroblock is 24 (equal to 96 clock cycles). Table 5 shows the data flow of the pipeline processing approach.

5. Result

The simulation results are shown in Table 6. The architecture of SPA as a co-processor can accelerate a H.264/AVC decoder system. Moreover, the number of total memory references for load and store is reduced by 34% and 36% re-

Table 6 The performance comparison.

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

spectively.

5.1 Memory Performance

As the ITU-T Recommendation [1] and previous proposal [24]–[30] and [31] suggest, an adaptive filtering shall be applied to all 4×4 block edges of a picture. Most of the 4×4 blocks need to be filtered 4 times with the adjacent blocks (left, right, top, and bottom), except for the macroblock at boundary of a picture. Therefore, the number of total memory reference for each macroblock, including read and write, is $4 \times 4 \times 2 \times 16 = 512$ (Assume that the memory bus width is 32 bits).

In our proposed SPA architecture with the presented processing order, the memory performance is improved by 4 times, when compared to software implementation. Table 7 shows the comparison of various architectures. Our SPA architecture reduces the memory access times from 592 to 192, when compared to the previous design in [24].

5.2 System Performance

As described in previous section, using the proposed processing order, the total filtering for one luma and two chroma macroblocks takes 96 (24×4) and $32(8 \times 4) \times 2 = 64$ cycles respectively. As a result, the total filtering takes 160 cycles for a luma and two chroma macroblocks. Our filtering scheme takes less number of cycles when compared to 240, 286, 240, and 192 cycles of the architecture described in [24], [26], [28] and [31]. Table 8 shows the performance comparison of various architectures. The cycle counts of loads and stores between the external and internal memory are not calculated for a fair comparison.

5.3 Implementation

We implemented the SPA architecture by Verilog HDL and synthesized the design using TSMC 0.18 μm Artisan CMOS cell library using Synopsys Design Compiler with critical path constraint set to 5 ns (200 MHz). The synthesized gate count is shown in Table 9 for edge filter unit and Table 10 for SPA architecture.

Another cost effective architecture is SPA-FIFO architecture. It can save 6088 gate counts when we use memory FIFO instead of register array to implement the Ship Operation Array (Array1, Array3, Array4, Array5, Array6, and Array7). The hardware comparison of various architectures is shown in Table 11. Although the SPA-FIFO architecture introduces some hardware overhead due to transposing circuit and computation of boundary strength, the implementation is also suitable for embedded systems. This is because the cache memory and memory buffer occupy most of area in a MPEG-4 codec system.

Table 7 Memory reference per macroblock.

Author	Architecture	MEM
JM9.2 [1]	Software Implementation	768
Huang [24]	Basic+Single-port SRAM	768
Huang [24]	Advance+Dual-port SRAM	384
Huang [24]	Basic+Two-port SRAM	768
Huang [24]	Dual Arrays+Two-port SRAM	384
Chen [28]	Dual-port or Two Single port SRAM	192
Li [31]	5120 bits Dual-Port SRAM	192
SPA	Dual-port or Two Single port SRAM	192

Table 8 Processing cycles per macroblock.

Author	Architecture	Cycl
Huang [24]	Basic+Single-port SRAM	504
Huang [24]	Advance+Dual-port SRAM	440
Huang [24]	Basic+Two-port SRAM	408
Huang [24]	Dual Arrays+Two-port SRAM	240
Sheng [26]	2-D Deblocking Filter	286
Chen [28]	Dual-port or Two Single port SRAM	240
Li [31]	5120 bits Dual-Port SRAM	192
SPA	Dual-port or Two Single port SRAM	160

Table 9 The area of edge filtering unit.

Item	Function	Gate
1.	Alpha Table derived	137
2.	Beta Table derived	87
3.	CLIP Table	66
4.	Luma4 Filtering Operation	1372
5.	Chroma4 Filtering Operation	247
6.	Standard Filtering for Luma and Chroma	811
7.	Content Activity Check	1937
8.	Boundary Strength	1847
Total	Edge Filter Unit	6300

Table 10 The area of SPA architecture.

Item	Function	Gate Count
1.	Horizontal Filtering Unit	6300
2.	Vertical Filtering Unit	6300
3.	SPA Processing Circuit	8247
Total	SPA Architecture	20847

Table 11 The hardware comparison of various architectures.

Author	Architecture	Gate Count
Huang [24]	Basic+Single-port SRAM	18.91 K
Huang [24]	Advance+Dual-port SRAM	20.66 K
Li [31]	5120 bits Dual-Port SRAM	9.57 K
Chen [27]	Dual-port SRAM	5.66 K
Chen [28]	Dual-port SRAM	22 K
SPA	Dual-port SRAM	20.84 K
SPA-FIFO	Dual-port SRAM	14.75 K

6. Conclusion

In this paper, we propose an efficient 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 SPA using a novel processing order. As a result, the SPA can be used in a high performance system which only requires a simple bus interface for the integration into video SoC platforms that support a wide range of applications such as video telephone, video conferencing, video streaming, digital video authoring, and many others.

Acknowledgments

The work in this paper is in part supported by the National Science Council, Taiwan R.O.C., under NSC 94-2220-E-006-004 and NSC 94-2220-E-006-008.

References

[1] ITU-T Recommendation H.264, "Advanced video coding for generic audiovisual services," March 2003.
[2] ITU-T Recommendation H.261, "Video codec for audiovisual services at p X 64 kbit/s," March 1993.
[3] ISO/IEC 11172, "Information technology: Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s," Geneva, 1993.
[4] ISO/IEC 13818-2, "Generic coding of moving pictures and associated audio information-Part 2: Video also ITU-T Recommendation H.262," 1994.
[5] ITU-T Recommend H.263, "Video coding for low bit rate communication," 1998.
[6] ISO/IEC 14496-2, "Information technology: Coding of audiovisual objects-Part 2: visual," Geneva, 2000.
[7] P. List, A. Joch, J. Lainema, G. Bjøntegaard, and M. Karczewicz, "Adaptive deblocking filter," IEEE Trans. Circuits Syst. Video Technol., vol.13, no.7, pp.614–619, 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, July 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.1, 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, July 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, July 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, July 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, July 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, July 2003.
[16] B. Girod, M. Kalman, Y.J. Liang, and R. Zhang, "Advances in video channel-adaptive streaming," Proc. ICIP 2002, pp.9–12, Rochester, NY, Sept. 2002.
[17] Y.J. Liang and B. Girod, "Low-latency streaming of pre-encoded video using channel-adaptive bitstream assembly," Proc. ICME 2002, pp.873–876, Lausanne, Switzerland, Aug. 2002.
[18] S.H. Kang and A. Zakhor, "Packet scheduling algorithm for wireless video streaming," Proc. International Packet Video Workshop, pp.1–11, Pittsburgh, PA, April 2002.
[19] S. Wenger, "H.264/AVC over IP," IEEE Trans. Circuits Syst., vol.13, no.7, pp.645–656, July 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, July 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 Circuit Syst. Mag., vol.4, no.1, pp.7–28, 2004.
[22] M.I.T. Orchard and G.J. Sullivan, "Overlapped block motion compensation: An estimation-theoretic approach," IEEE Trans. Image Process., vol.3, no.5, pp.693–699, 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, 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," Proc. IEEE Conf. Multimedia and Expo, pp.693–696, 2003.
[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, 2004.
[26] B. Sheng, W. Gao, and D. Wu, "An implemented architecture of deblocking filter for H.264/AVC," IEEE International Conference on Image Processing (ICIP'04), vol.1, pp.665–668, Oct. 2004.
[27] C.-M. Chen and C.-H. Chen, "An efficient VLSI architecture of edge filtering in H.264/AVC," IASTED International Conf. on Circuits, Signals, and Systems, pp.118–122, Oct. 2005.
[28] C.-M. Chen and C.-H. Chen, "An efficient architecture for deblocking filter in H.264/AVC video coding," IASTED International Conf. on Computer Graphics and Imaging, pp.177–181, Aug. 2005.
[29] C.-M. Chen and C.-H. Chen, "Parallel processing for deblocking filter in H.264/AVC," IASTED International Conf. on Communications, Internet, and Information Technology, pp.188–191, Oct. 2005.
[30] C.-M. Chen and C.-H. Chen, "A memory efficient VLSI architecture for deblocking filter in H.264 using vertical processing order," IEEE International Conf. on Intelligent Sensors, Sensor Networks and Information Processing, pp.361–366, Dec. 2005.
[31] 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, July 2005.
[32] D.C. Burger and T.M. Austin, "The SimpleScalar tool set, Version 2.0," University of Wisconsin, Madison Technical Report, 1997.
[33] The 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 in 1996. Currently, he is a Ph.D. candidate in the department of Electrical Engineering, National Cheng Kung University. He joined Elan, Corp. (Fabless), Hsin-Chu, Taiwan, R.O.C., in 2000, where he is currently Deputy Manager for the department of system chip design division. His research interests include computer architecture, VLSI architecture of video-coding, DSP architecture design, and Network Processor.

Chung-Ho Chen received his MSEE degree in electrical engineering from the University of Missouri-Rolla in 1989. He received the Ph.D. degree in electrical engineering from the University of Washington, Seattle, in 1993. Since 1993, he was the 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, where he is currently an Associate Professor. His research areas include advanced computer architecture, video technology, and network storages. He holds a U.S. patent on a multi-computer 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.