Proceedings of the Fourth IASTED International Conference
Communications, Internet, and Information Technology
October 31 - November 2, 2005, Cambridge, USA

ALTERNATIVE PROCESSING ORDER WITH EFFICIENT ARCHITECTURE FOR ADAPTIVE DEBLOCKING FILTER IN H.264/AVC

Chung-Ming Chen and Chung-Ho Chen
Department of Electrical Engineering &
Institute of Computer and Communication Engineering
National Cheng Kung University
Taiwan, R.O.C.
cmchen@casmail.ee.ncku.edu.tw

Jian-Ping Zeng, Yu-Pin Chang, and Jing-Jou Tang
Department of Electric Engineering
Southern Taiwan University of Technology
Taiwan, R.O.C.
m9430107@webmail.stut.edu.tw
ypchang@mail.stut.edu.tw

ABSTRACT
In this paper, we study and analyze the memory reference of deblocking filter in H.264/AVC baseline decoder based on SimpleScalar/ARM simulator. In order to reduce the number of memory references and thus improve overall system performance in an embedded system, we propose an advanced filtering process order with an efficient VLSI architecture which simultaneously processes the horizontal filtering of vertical edge and vertical filtering of horizontal edge. As a result, the performance of the proposed scheme is 129% faster than the advanced architecture of a previous proposal. Moreover, the number of the total memory references is reduced by 78.75% and 52.5% respectively compared to the basic and advanced architectures of the previous works.

KEY WORDS
Deblocking Filter, H.264/AVC, Video Coding.

1. Introduction

The H.264/AVC is the latest video coding standard in a series of such standards: H.261, MPEG-1, MPEG-2, H.263 [1], and MPEG4. It was approved by the ITU-T as Recommendation H.264 [2] and by ISO/IEC as MPEG-4 part 10, Advance Video Coding (AVC) [3] in May 2003. The functional blocks of H.264/AVC, as well as their features, are shown in Figure 1.

As shown in the previous study [4], the most time consuming portion of H.264/AVC video decoder is the deblocking filter. It can be separated into two sub-functions: the computation of the "boundary strength" parameter, Bs for each 4-samples and the content-dependent filtering process. Due to intensive computations, in [5] and [6] dedicated hardware was developed to accelerate only the content-dependent filtering process. In this work, we present an advanced processing method with an efficient VLSI architecture which implements both the computation of boundary strength parameter and content-dependent filtering process. This new design significantly outperforms the previous proposals in [5] and [6].

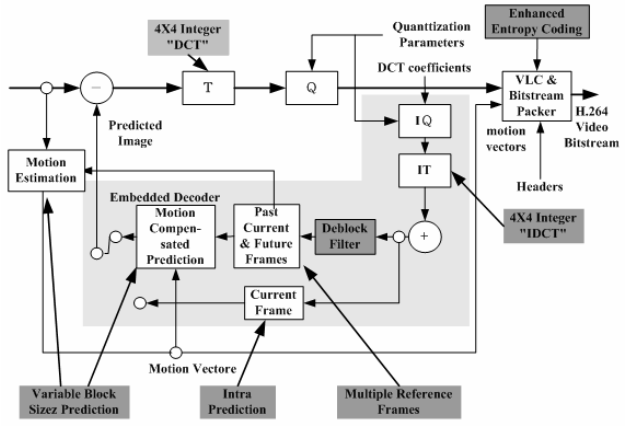

Figure 1: Block Diagram of H.264/AVC

The organization of this paper is as follows. In Section 2, the algorithm of the deblocking filter is explained. Section 3 illustrates the block diagram of our previous proposed architectures in [7], [8] and an advanced processing order for this paper. Section 4 shows the simulation result. Finally, conclusion is presented in Section 5.

2. Algorithm of Deblocking Filter

On the sample processing level, content of a set of sample and quantization parameter threshold can turn on/off the filtering for each individual sample. For example, as shown in Figure 2, whether the samples p0 and q0 as well as p1 and q1 are filtered is determined by using the quantization parameter (QP), dependent threshold Alpha(QP), and Beta(QP). Thus filtering of p0 and q0 only takes place if the following content activity check is satisfied:

1. Bs != 0
2. |p0 - q0| < Alpha(QP)
3. |p1 - p0|< Beta(QP) and |q1 - q0| < Beta(QP).

Correspondingly, filtering of p1 or q1 takes place if the condition below is satisfied:

$$|p2-p0| < Beta(QP) \text{ or } |q2-q0| < Beta(QP).$$

A detailed description of the adaptive deblocking filter can be found in [9]

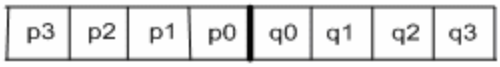

Figure 2: Samples Across a 4x4 Block

3. Architecture

3.1 Edge Filter Operation

The complexity of H.264/AVC deblocking filter is mainly based on two reasons. The first reason is the high adaptive filtering, which requires several conditional processing on each block edges and sample levels. As described in the previous section, the threshold value of Alpha and Beta, the table-derived operations, and edge filtering operations are known to be very time consuming. Therefore, we implemented an efficient VLSI architecture that includes content activity checks, the table-derived operations, and filter operations into edge filter unit to accelerate the horizontal and vertical filtering on the boundary of two adjacent basic 4x4 blocks as shown in Figure 3. A detailed description of the edge filtering unit can be found in [7].

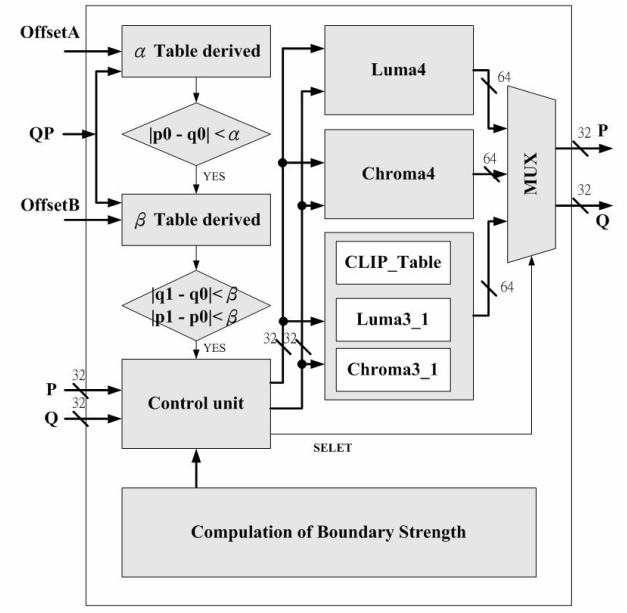

Figure 3: Edge Filtering Unit

3.2 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. Each sample in a picture must be loaded/stored from/to memory 4 times; either to be modified or to determine if the neighboring samples will be modified. In order to reduce the numbers of memory reference and improve the overall system performance, we proposed another efficient architecture, which can simultaneously processes the horizontal filtering of vertical edge and vertical filtering of horizontal edge as shown in Figure 4. The proposed architecture is called "Pipeline Buffer Shift Registers (PBSR)".

There are three major functions in our architecture [8]. The first component is the Shift Operation Array. There are eight forwarding shift register arrays in the architecture (for example, Array1, 2, 4, 5, 6, 7, 8, and 9). Each array contains four entries which contains 4 processed samples. The shift direction is shown in Figure 4. The second function of our proposed architecture is the transposing operation as shown in Figure 4. The Array3 and Array10 latched the 4x4 block sample values that are transposed from Array 2 and Array9 respectively. And the final important functions are the horizontal and vertical filter units which are described in the previous subsection.

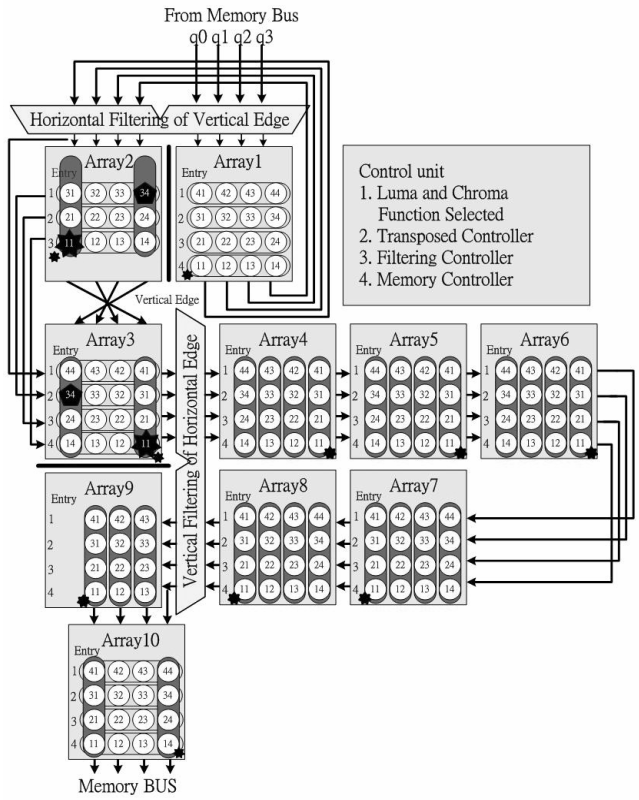

Figure 4: Proposed Architecture

3.3 Basic Data Flow

In this subsection, the basic processing method using raster scan order for a picture is shown in Figure 5. The data flow for basic processing order to process each macroblock is presented in Table 1. For a basic 4x4-block, it takes 13 block cycles (52 clock cycles) to process the

first block B1 and the number of total processing time for each macroblock is 32 block cycles (128 clock cycles). In the first 5 block cycles (the first 20 clock cycles), the blocks of E1, E2, E3, E4, and E5 are loaded from internal memory to PBSR's Array6, Array5, Arrar4, Array3, and Array1 respectively and no filtering operations are performed. In the next 2 block cycles (the sixth and seventh block cycles), the horizontal filtering of vertical edge V1 and V2 are performed respectively. In the eighth block cycle, the PBSR architecture can simultaneously process horizontal filtering of vertical edge V3 (the boundary of block B1 and B2) and vertical filtering of horizontal edge H3 (the boundary of block E1 and B1). Block E1 is written to the internal memory in next block cycle (the ninth block cycle). In the thirteenth block cycle, the vertical filtering of horizontal edges H7 (the boundary of block B1 and B5) are performed. At this time, block B1 has finished 4 times filtering with the adjacent blocks (left block E5, right block B2, top block E1, and bottom block B5). Finally, PBSR writes block B1 to memory at the fourteenth block cycle. Therefore, the number of total block cycles required for a QCIF picture is 32x99=3168 (12672 clock cycles).

Table 1: Basic Data Flow in the PBSR

State	Block Cycle							
	5	6	7	8	9	10	11	12
Array1	E5	B1	B2	B3	B4	E5	B5	B6
Array2	E4	E5	B1	B2	B3	B4	E5	B5
Array3	E4	E5	B1	B2	B3	B4	E5	B5
Array4	E3	E4	E5	B1	B2	B3	B4	E5
Array5	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
Array9				E1	E2	E3	E4	E5
Array10				E1	E2	E3	E4	E5
MEM					E1	E2	E3	E4

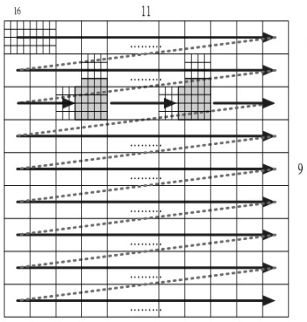

Figure 5: Basic Processing Order

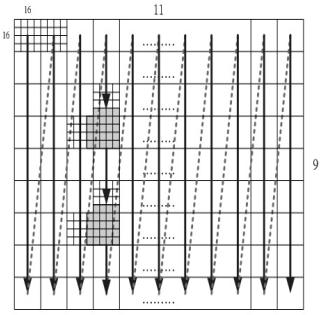

Figure 6: Advanced Processing Order

3.4 Advanced Data Flow

Based on our previous work, we present an improved processing order as shown in Figure 6. The vertical processing for a picture is in top-down order instead of raster scan order. The basic idea of using vertical processing is to buffer the initial blocks B13, B14, B15, and B16 in PBSR's Array8, Array7, Array6, and Array5 respectively as shown in Figure 7. When process a macroblock from M1 to M2, the PBSR with this advanced processing order does not need to load these initial blocks. And thus we can save 4 block cycles when process each macroblock, except for the macroblock at the bottom of a picture. As a result, the number of total block cycles to process a QCIF frame is 2021 (or 8084 clock cycles). Table 2 shows the data flow of the vertical processing order.

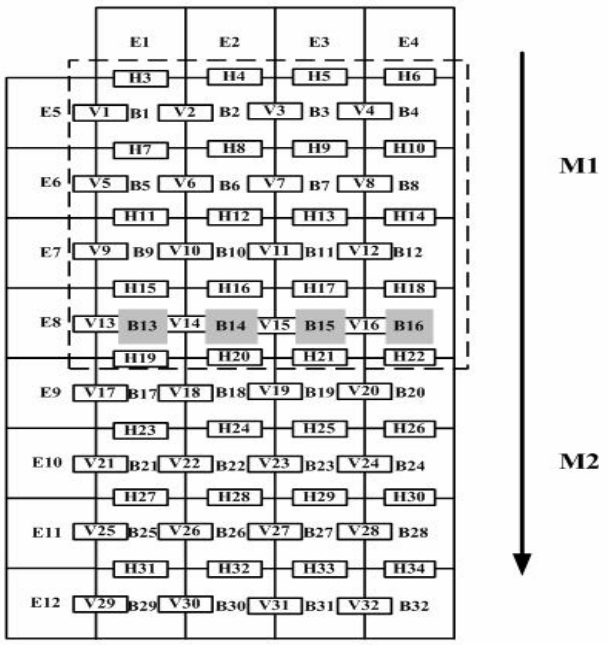

Figure 7: Processing Order

Table 2: Advanced Data Flow in the PBSR

State	Block Cycle							
	1	2	3	4	5	6	7	8
Array1	E9	B17	B18	B19	B20	E10	B21	B22
Array2	B16	E9	B17	B18	B19	B20	E10	B21
Array3	B16	E9	B17	B18	B19	B20	E10	B21
Array4	B15	B16	E9	B17	B18	B19	B20	E10
Array5	B14	B15	B16	E9	B17	B18	B19	B20
Array6	B13	B14	B15	B16	E9	B17	B18	B19
Array7		B13	B14	B15	B16	E9	B17	B18
Array8			B13	B14	B15	B16	E9	B17
Array9				B13	B14	B15	B16	E9
Array10				B13	B14	B15	B16	E9
MEM					B13	B14	B15	B16

4. Result

The simulator used in this study is derived from the SimpleScalar/ARM tool set [10], a suite of functional and timing simulation tools for ARM ISA. Our baseline

simulation configuration models the Intel's StrongARM SA-110 processor.

4.1 Memory Reference

Table 3 shows the comparison of various architectures for the number of memory references. Our architecture PBSR using the advanced processing order can save 78.75% and 52.5% of the memory bandwidth respectively, when compared to the previous schemes in [5]. Hence, our architecture is able to significantly reduce power consumption in an embedded system.

Table 3: Memory References per Macroblock

Author	Architecture	MEM Ref
JM9.2 [2]	Software Implementation	768
Huang [5]	Basic + Single-port SRAM	768
Huang [5]	Advanced + Dual-port SRAM	384
Huang [5]	Basic + Two-port SRAM	768
Huang [5]	Dual Arrays + Two-port SRAM	384
Basic PBSR	Dual-port SRAM or Two Single port SRAM	192
Advanced PBSR	Dual-port SRAM or Two Single port SRAM	176

4.2 Performance

Using the advanced processing order, the total number of filtering cycles for one luma and two chroma macroblocks takes 20x4=80 and (6x4=24)x2=48 respectively. As a result, the filtering takes 128 cycles for a luma and two chroma macroblocks. Our filtering scheme takes less number of cycles when compared to 294 cycles of the architecture described in [5]. Table 4 shows the performance comparison of various architectures.

Table 4: Memory Cycles per MB

Author	Architecture	Cycles MB
Huang [5]	Basic + Single-port SRAM	558
Huang [5]	Advanced + Dual-port SRAM	494
Huang [5]	Basic + Two-port SRAM	462
Huang [5]	Dual Arrays + Two-port SRAM	294
Basic PBSR	Dual-port SRAM or Two Single port SRAM	240
Advanced PBSR	Dual-port SRAM or Two Single port SRAM	128

5. Conclusion

In this paper, we study and analyze the memory reference of deblocking filter in H.264/AVC baseline decoder based on SimpleScalar/ARM simulator. In order to reduce the number of memory references, we propose an advanced processing order with an efficient VLSI architecture which simultaneously processes the horizontal filtering of vertical edge and vertical filtering of horizontal edge. Simulation results show that the processing capability of our proposed architecture is very appropriate for real-time deblocking of 1280x720 30Hz video operating at 100MHz. According to the simulation results, this design is a good option to use for the deblocking filter in an embedded system design with H.264/AVC coding systems.

Acknowledgements

The work in this paper is in part supported by the National Science Council, Taiwan ROC, under NSC 93-2220-E-006-004. In addition, the authors thank Elan Microelectronics CORP (Fabless Semiconductor Corp.). for support in VLSI design flow and simulation environment.

References

[1] ITU-T Recommend H.263, *Video Coding for Low Bit Rate Communication*, 1998.

[2] ITU-T Recommendation H.264, *Advanced video coding for generic audiovisual services*, 2003.

[3] ISO/IEC 14496–10:2003, *Coding of Audiovisual Objects—Part 10: Advanced Video Coding*, 2003.

[4] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro, H.264/AVC baseline profile decoder complexity analysis, *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, 2003, 715–727.

[5] Yu-Wen Huang, To-Wei Chen, Bing-Yu Hsieh, Tu-Chih Wang, Te-Hao Chang, and Liang-Gee Chen, "Architecture Design for De-blocking Filter in H.264/JVT/AVC," *Proc. IEEE Conf. on Multimedia and Expo*, 2003, 693-696.

[6] Miao Sima, Yuanhua Zhou, and Wei Zhang, "An Efficient Architecture for Adaptive Deblock filter of H.264/AVC Video Coding," *IEEE Transactions on Consumer Electronics*, Vol. 50, 2004, 292-296.

[7] Chung-Ming Chen, Chung-Ho Chen, "An Efficient VLSI Architecture for Edge Filtering in H.264/AVC," *IASTED CSS*, 2005.

[8] Chung-Ming Chen, Chung-Ho Chen, "An Efficient Architecture for Deblocking Filter in H.264/AVC Video Coding," *IASTED CIGM*, 2005.

[9] Peter List, Anthony Joch, Jani Lainema, Gisle Bjøntegaard, and Marta Karczewicz, Adaptive Deblocking Filter, *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, 2003, 614-619.

[10] Douglas C. Burger and Todd M. Austin, The SimpleScalar Tool Set, Version 2.0. University of Wisconsin, Madison Tech. Report. 1997.