AN EFFICIENT DEBLOCKING FILTER WITH SELF-TRANSPOSING MEMORY ARCHITECTURE FOR H.264/AVC

Mahdi Nazm Bojnordi, Omid Fatemi, Mahmoud Reza Hashemi
School of Electrical and Computer Engineering, University of Tehran, Tehran, IRAN.
Email: m.bojnordi@ece.ut.ac.ir, omid@fatemi.net, hashemi@comnete.com

ABSTRACT

One of the main reasons behind the superior efficiency of the H.264/AVC video coding standard is the use of an in-loop deblocking filter. Since the deblocking filter is computation and data intensive, it has a profound impact on the speed degradation of both encoding and decoding processes. In this paper, we propose an efficient deblocking filter architecture that can be used as an IP core either in the dedicated or platform-based H.264/AVC codec systems. Novel self-transposing memory unit is used in this paper to alleviate switching between the horizontal and vertical filtering modes. Moreover, to reduce the processing latency, a two-stage pipelined architecture is designed for 1-D filter that produces output data after 2 clock cycles. With a clock of 100 MHz the proposed design is able to process a 1280×1024 (4:2:0) video at 25 frame/second. The proposed architecture offers 33% to 56% performance improvement compared to the existing state-of-the-art architectures.

1. INTRODUCTION

Most video coding techniques employ block-based prediction, transformation, and quantization for encoding. The use of these block-based tools, however, decreases inter-block correlation in video frames and adds visible blocking structures to the reconstructed frame, i.e. blocking artifacts [1]-[3]. The newest video coding standard, H.264/AVC [4], uses an in-loop adaptive filter to eliminate the blocking artifacts. Among various coding tools of the H.264/AVC codec, the in-loop deblocking filtering module has a profound impact on the video visual quality improvement [5]. However this improvement is achieved at the cost of large amount of computation and memory read/write operations. Therefore its computational complexity significantly reduces the encoding/decoding speed. The deblocking filter is described in detail in [6]. The advantages of the in-loop deblocking over post filters are also discussed in this reference. As shown in [1], the in-loop deblocking filter reduces the bit-rate typically by 5%-10% preserving the same objective video quality.

Recently, there are many proposed architectures for H.264/AVC deblocking filter [7]-[11]. Generally, in order to design an efficient real-time architecture for H.264/AVC deblocking filter, two major issues should be addressed, including 1-D filter processing latency and memory data access.

In this paper, a high-performance low-cost deblocking algorithm is proposed. Using an efficient memory management and self-transposing memory architecture, performance improvement of up to 56% is achieved, compared to the existing state-of-the-art architectures. The paper is organized as follows. An introduction to the H.264/AVC deblocking algorithm is given in Section 2. In Section 3, our proposed architecture is described. Comparison and experimental results are presented in Section 4. Finally, the paper ends with a conclusion Section 5.

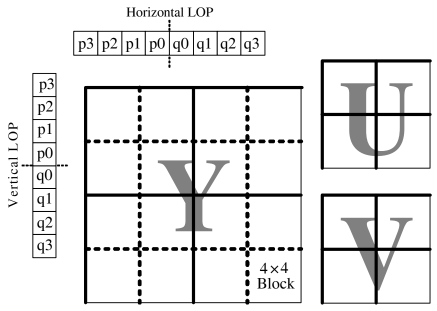

Fig. 1- Macroblock boundaries that should be filtered in each of the three Y, U, and V components.

2. THE H.264/AVC DEBLOCKING ALGORITHM

According to the latest H.264/AVC Recommendation [4], the deblocking filtering algorithm is defined as a conditional process that has to be applied to vertical and horizontal edges of all N×N blocks of each macroblock. For luminance component (Y) N is selected equal to the size of the applied I-transform, i.e. 4 or 8. For chrominance components (U and V) N is selected equal to 4. Fig. 1 shows the boundaries inside a macroblock that have to be filtered. The boundaries indicated by bold lines should be filtered for both sizes of applied transform. However, the edges indicated by doted lines need to be filtered only if 4×4 transform is applied to the desired macroblock.

In order to filter both vertical and horizontal block boundaries, the algorithm processes data in the form of horizontal and vertical line of pixels (LOP). Each LOP is composed of 8 neighboring pixels between two 4×4 blocks. Vertical and Horizontal LOPs are illustrated in Fig. 1. In each macroblock, the horizontal LOPs are filtered first. Then the vertical LOPs are filtered. In fact, the H.264/AVC deblocking filtering is a procedure for updating the content of LOPs using of a set of adaptive low-pass filters. For each edge, the appropriate filter is selected from the filter bank according to the boundary strength (*bS*), thresholds α and β and content of the LOP. For each two neighboring blocks (*p* and *q*), the *bS* factor is determined according to Table 1. α and β thresholds depend on the quantization parameter (*QP*) and some other syntax elements. Based on these parameters, and for each LOP the algorithm determines whether or not filtering is required. If all the following conditions are true, the filter will be applied to the desired LOP:

$$bS \neq 0,\ |p0 - q0| < \alpha,\ |p1 - p0| < \beta,\ |q1 - q0| < \beta \cdot$$

Table 1- Decision flow for determining the *bS* value.

Condition	*bS* value
p or q is intra coded and boundary is a macroblock boundary	4
p or q is intra coded and boundary is not a macroblock boundary	3
neither p or q is intra coded; p or q contain coded coefficients	2
neither p or q is intra coded; neither p or q contain coded coefficients; p and q have different reference frames or a different number of reference frames or different motion vector values	1
neither p or q is intra coded; neither p or q contain coded coefficients; p and q have same reference frame and identical motion vectors	0

3. PROPOSED DEBLOCKING FILTER ARCHITECTURE

H.264/AVC presents a macroblock-based algorithm for deblocking the reconstructed frames. Implementing the algorithm as described there needs big buffers and circuitry for maintaining and transposing all the macroblock data. It is resulted in both hardware complexity and performance degradation. Dividing the video frames into smaller blocks alleviates design complexity. This paper proposes an 8×8 block based architecture for deblocking filtering in H.264/AVC. Processing data elements are chosen 8×8 box of pixels (BOP) accommodating 8 vertical/horizontal LOPs. Block diagram of the proposed architecture is shown in Fig. 2 (a). The architecture is composed of three major parts: *LOP-filter*, *Memory Unit*, and *Controller*.
As illustrated in Fig. 2 (b), for each BOP we have to filter four edges. v_0 and h_0 represent the boundaries indicated by doted lines in Fig. 1. These edges are skipped during deblocking filtering of luminance component if the 8×8 transform is chosen.

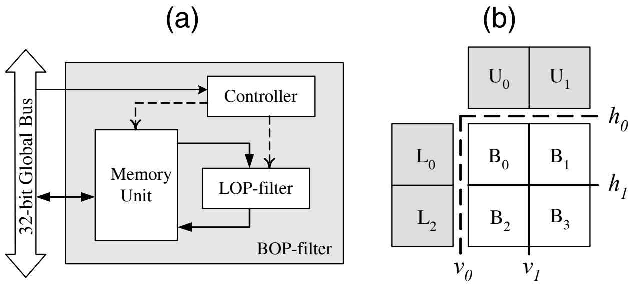

Fig. 2- The BOP-filter architecture (a); Boundaries that should be filtered in each BOP (b).

Three parts of the BOP-filter are described in the following:

3.1. LOP-filter Architecture

The most complex part of the BOP-filter is the one-dimensional LOP-filter. The maximum latency of the total design is caused by this module. This module consumes two clock cycles for preparing the filtered result of an 8-pixel input. It can be used for both *horizontal* and *vertical* LOP filtering modes. Inside this module the *bS* factor and filtering thresholds are evaluated using the parameters specified by the encoding/decoding parts. The determined parameters and content of the LOP are used for updating the new values for the LOP pixels. The updated LOP values will be presented at the LOP-filter output after two clock cycles. According to the H.264/AVC deblocking algorithm, for each input 8-pixel LOP, at most six pixels should be updated.

3.2. Self-Transposing Memory Architecture

As mentioned above the new deblocking scheme requires a new data memory organization. BOP data are considered in the form of pairs of pixels in the memory. Each two neighboring pixels compose a 16-bit word in the memory. Therefore, each BOP is mapped in the memory as depicted in Fig. 3. According to this memory mapping scheme, the proposed self-transposing memory architecture needs two ports: one for data read and another for writing. Data words of size 16-bit are accessed in this memory. Consequently, reading or writing each LOP needs four clock cycles. The BOP-filter employs this architecture beside the LOP-filter module resulted to less implementation cost. The memory structure is presented in more details in Fig. 4. This module uses two banks of memory modules to accommodate two BOPs inside. Each bank is composed of two *odd* and *even* 8-bit dual-port SRAMs.

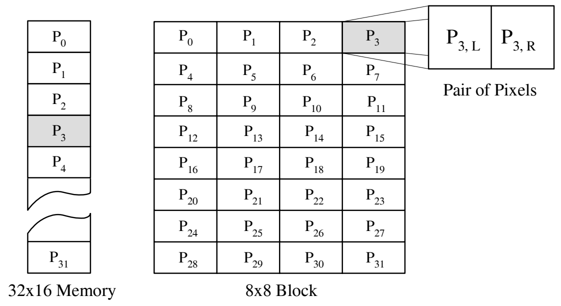

Fig. 3- BOP memory map.

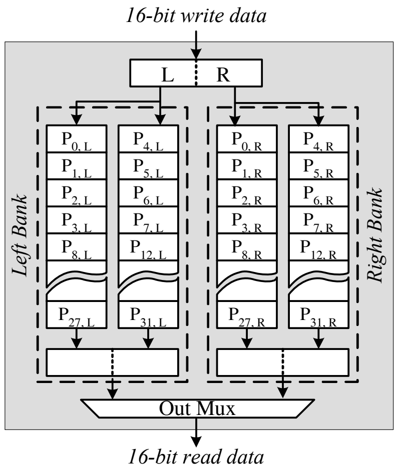

Fig. 4- Self-transposing memory architecture.

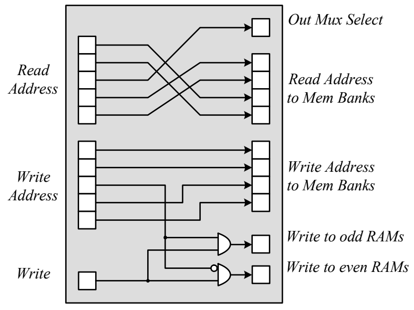

Fig. 5- Simple controller architecture for self-transposing memory.

The 16-bit input data is written in the form of two right and left 8-bit data in the right and left banks, respectively. In other words, no transposition occurs during writing a 16-bit data in the memory banks. However, reading mechanism is modified in such a way to capture transposed LOP from the output data port. A simple controller is designed that controls writing and reading data to/from the memory banks. Control signals to the memory banks are generated simply by reordering the input write and read address lines. Fig. 5 shows the controller structure for the proposed self-transposing memory architecture.

Common read and write addresses are generated for all the SRAM modules. Memory write operation is controlled by the write signal and input write address. If the address value points to the pixels in an odd row, the input data should be written in the odd RAM modules of both left and right banks. Otherwise the input data has to be written in the even RAM modules. Reading odd or even columns of the BOP is controlled by the value of the read address.

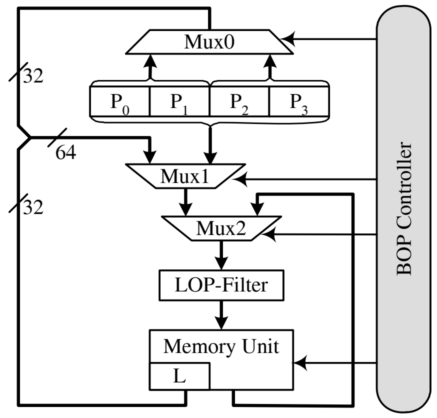

Fig. 6- BOP internal dataflow diagram.

3.3 BOP-filter Controller

The BOP controller module is responsible for controlling all the functions of the BOP-filter module. The new BOP is read from the global data bus at the horizontal filtering mode. Assuming a typical 32-bit system bus, every two clock cycles the 64-bit data is fed to the LOP-filter. In the horizontal filtering mode, newly read data from the bus and the previously stored data in the local memory are merged and fed to the LOP-filter. The updated LOP will be stored in the local memory. Once all the horizontal LOPs belonging to the desired BOP are read and filtered, BOP-filter goes into the vertical filtering mode. Operating at the vertical filtering mode, data are read from self-transposing memory and merged with the previously stored data to pass to the LOP-filter. The output data are generated during the filtering operation and can be stored in the destination frame memory. In fact our strategy is based on the delayed data write back mechanism. After filtering each BOP the grey data blocks that are shown in Fig. 2 (a) will be written back into the memory. In other words, left and right nibbles of each LOP should be processed with their left- and right-hand side nibbles. Therefore, each nibble can be written back to the memory if its two filtering processes are performed. Fig. 6 shows internal data flow of the proposed BOP-filter. As shown in this figure, input data to the LOP-filter is selected from the newly loaded data from the global bus and previously stored blocks in the memory (named L bank). According to the selected edge for filtering in the

4×4 mode, *Mux0* selects between the left and right half of the newly read LOP. Output of this multiplexer is concatenated to the data comes from L bank and composes a LOP for filtering the 4×4 mode. If the filtering mode is 8×8, the newly read LOP should be passed to the LOP-filter. *Mux1* selects the proper data for according to applied transform. Finally, distinguishing between the horizontal and vertical filtering is performed using *Mux2*. The 32-bit right half of LOP-filter result is stored in the L bank to be used later. It is guaranteed with that the updated LOP be ready after at most four clock cycles either if the transform size of 8 or 4 chosen. After filtering v_1 the LOPs are stored in the transposing memory. Once all the horizontal LOPs are filtered and stored in the transposing memory, BOP transpose should be processed for filtering horizontal edges. This memory unit is designed to transpose the BOP with no need to extra clock cycles. The BOP can be transposed only by writing/reading to/from this memory unit.

Table 2- Comparison between the proposed deblocking filters and other architectures.

	[11] A	[11] B	[11] C	[11] D	[10]	proposed
Gate # (K)	18.91	18.91	18.91	20.66	9.35	8.24
Max Cycles/MB	878	814	782	614	566	384

4. COMPARISON AND EXPERIMENTAL RESULTS

The proposed architecture is implemented with Verilog synthesized by a standard 0.35μ CMOS technology library. Synthesis results indicate that delay of the critical path is 8.63 ns. Therefore, the proposed deblocking filter can work with a clock frequency of 100MHz. At this clock rate, the proposed architecture can process at most 50 Mega-LOPs per second. In terms of BOPs, the architecture is able to perform deblocking of each BOP during 64 clock cycles. Considering the clock frequency of 100MHz it can perform deblocking of 1600 Kilo-BOPs per second. Four different deblocking architectures are proposed in [11]. The architectures were designed to support macroblock based filtering in two *basic* and *advanced* modes According to the memory modules used in each of them, four architectures are categorized as (A) basic + single port SRAM, (B) advanced + dual port SRAM, (C) basic + dual port SRAM, and (D) advanced with dual register array + dual port SRAM. Another architecture with a 2-D memory organization is also proposed in [10]. Results of comparison with these architectures are summarized in Table 2. The number of gates provided in this table excludes SRAM modules. As illustrated, the proposed architecture offers 33% to 56% performance improvement compared to the existing architectures.

In contrast to the other existing implementations for deblocking filtering, the proposed architecture uses RAM banks that are less complex than the register files and two-dimensional memory used in [10]. Also we have used an interleaved memory access for reading and writing data on the global bus. It utilizes only 50% of the bus operation during the filtering process.

5. CONCLUSIONS

In this paper, a cost efficient deblocking architecture is proposed that can be used in both dedicated fully pipelined architectures and platform based H.264/AVC codecs. Using a novel self-transposing memory architecture and extra memory banks for maintaining previously filtered blocks, memory data accesses are optimized. Also the design supports filtering of both horizontal and vertical edges with no modification in the filtering mechanism

6. ACKNOWLEDGMENT

The authors wish to express their gratitude to Iran Telecommunication Research Center (ITRC) for their financial support during the course of this research.

7. REFERENCES

[1] T. Wiegand, G.J. Sullivan, G. Bjontegaard, A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 13, no.7, pp. 560-576, Jul. 2003.

[2] S. D. Kim, J. Yi, H.M. Kim, J.B. Ra, "A Deblocking Filter with Two Separate Modes in Block-Based Video Coding," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 9, no. 1, pp. 156- 160, Feb. 1999.

[3] R. Schäfer, T. Wiegand, H. Schwarz, "The Emerging H.264 /AVC Standard," *EBU Technical Review*, Jan. 2003.

[4] "Advanced video coding for generic audiovisual services," *ITU-T Rec. H.264/ISO/IEC 14496-10*, Mar. 2005.

[5] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, T. Wedi, "Video coding with H.264/AVC: tools, performance, and complexity," *IEEE Circuits and Systems Magazine*, vol. 4, iss. 1, 2004.

[6] P. List, A. Joch, J. Lainema, G. Bjontegaard, M. Karczewicz, "Adaptive deblocking filter," *IEEE Trans. Circuits Systems for.Video Technology.*, vol. 13, pp. 614- 619, July 2003.

[7] B. Sheng, W. Gao, D. Wu, "An implemented architecture of deblocking filter for H.264/AVC," *Proc. of Image Processing (ICIP)*, vol. 1, pp. 665 - 668, Oct. 2004.

[8] M. Sima, Y. Zhou, W. Zhang, "An Efficient Architecture for Adaptive Deblocking Filter of H.264/AVC Video Coding," *IEEE Trans. on Consumer Electronics*, vol. 50, no. 1, pp. 292- 296, Feb. 2004.

[9] T. Liu, W. Lee, T. Lin, C. Lee, "A Memory-Efficient Deblocking Filter for H.264/Avc Video Coding," *IEEE International Symposium on Circuits and Systems*, pp. 2140 - 2143, 2005.

[10] L. Li, S. Goto, T. Ikenaga, "An Efficient Deblocking Filter Architecture with 2-Dimensional Parallel Memory for H.264/AVC", *Proc. of Asia South Pacific Design Automation Conference (ASP-DAC)*, pp. 623-625, Jan., China, 2005.

[11] Y.W. Huang, T.W. Chen, B.Y. Hsieh, T.C. Wang, T.H. Chang, L.G. Chen, "Architecture Design for De-blocking Filter in H.264/JVT/AVC", *Proc. of IEEE ICME 2003*, vol. 1, pp. 693-696, Jul. 2003.