AN IMPLEMENTED ARCHITECTURE OF DEBLOCKING FILTER FOR H.264/AVC

Bin Sheng[1] *Wen Gao*[1, 2, 3] *Di Wu*[1]

[1](Department of Computer Science and Technology, Harbin Institute of Technology, China)
[2](Institute of Computing Technology, Chinese Academy of Sciences, China)
[3](Graduate School of Chinese Academy of Sciences, China)
E-mail: {bsheng, wgao, dwu}@jdl.ac.cn

ABSTRACT

H.264/AVC is a new international standard for the compression of natural video images, in which a deblocking filter has been adopted to remove blocking artifacts. In this paper, we propose an efficient processing order for the deblocking filter, and present the VLSI architecture according to the order. Making good use of data dependence between neighboring 4x4 blocks, our design reduces the requirement of on-chip SRAM bandwidth and increases the throughput of the filter processing. The architecture has been described in Verilog HDL, simulated with VCS and synthesized using 0.25μm CMOS cells library by Synopsys Design Compiler. The circuit costs about 24k logic gates (not including a 32x64 SRAM and two 32x96 SRAMs) when the working frequency is set to 100MHz. This design can support real-time deblocking of HDTV (1280x720, 60fps) H.264/AVC video. This architecture is valuable for the hardware design of H.264/AVC CODEC.

1. INTRODUCTION

The Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG has finalized a new standard for the compression of natural video images. The new standard is known as H.264 and MPEG-4 Part 10, "Advanced Video Coding" [1]. H.264/AVC offers a significant improvement on coding efficiency compared to other compression standards such as MPEG-2. The functional blocks of H.264/AVC encoder and decoder are shown in Fig. 1 and Fig. 2 respectively.

The transformation algorithm adopted by H.264/AVC is the 4x4 Discrete Cosine Transform (DCT). In the DCT-based standards, such as H.263 [2], MPEG-2 [3] and MPEG-4 [4], annoying blocking artifacts arise when the compression ratio is high. This effect is caused by the non-overlapping nature of the block-based DCT coding and the quantization of DCT coefficients. Nowadays, there are two kinds of techniques to deal with the blocking artifacts, one of which is to reduce their occurrence and the other is to remove them.

H.263 and MPEG-4 adopt the overlapped block motion compensation (OBMC) [5] to reduce the blocking artifacts. Although OBMC works well with the blocking artifacts, its computation requirement of encoder motion estimation is much higher. Therefore, H.264/AVC adopts deblocking filter to remove blocking artifacts and to achieve much better subjective visual effects. The filter processing is applied after the inverse transformation and before reconstruction of the macroblock (MB) in both decoder and encoder, as shown in Fig. 1 and Fig. 2.

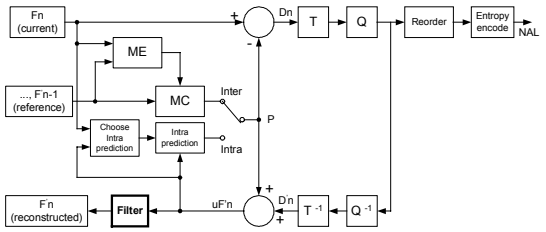

Fig. 1 H.264/AVC Encoder

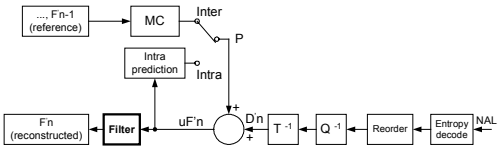

Fig. 2 H.264/AVC Decoder

Although some companies have developed their H.264/AVC CODEC such as Video Locus VLP4000 and MVI H.264 CODEC, they did not describe the implementation of deblocking filter in detail. Recently, Yuwen Huang gave a platform-based architecture, which can support real-time deblocking of 720p 30fps video [6]. Although using an "advanced processing order", which will be mentioned later, to reduce processing cycles, the architecture can't process fast enough to support real-time deblocking of HDTV (1280x720, 60fps) video at its maximum working frequency of 100MHz.

The remainder of the paper is organized as follows. In Section 2, algorithm of the deblocking filter is briefly explained. The implemented architecture is described in Section 3. Section 4 presents simulation results and VLSI implementation. Finally, conclusions are drawn in Section 5.

2. ALGORITHM

In H.264/AVC, the transformation is a 4x4-DCT, and the smallest block size of motion estimation is 4x4 too. So deblocking filter should be applied to all 4x4-block edges of a picture, except the edges at the boundary of the picture or any edges for which the deblocking filter is disabled by a special flag. According to H.264/AVC, the filtering is performed on MBs, one after another, with all MBs of a picture in raster scan order. The deblocking filter is invoked for luma and chroma components separately. For each MB, vertical edges are filtered from left to right, and horizontal edges are processed from top to bottom, as described in Fig. 3.

0-7803-8554-3/04/$20.00 ©2004 IEEE.

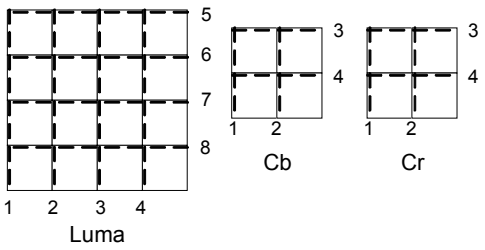

Fig. 3 Edges Filter Order in a 16x16 MB

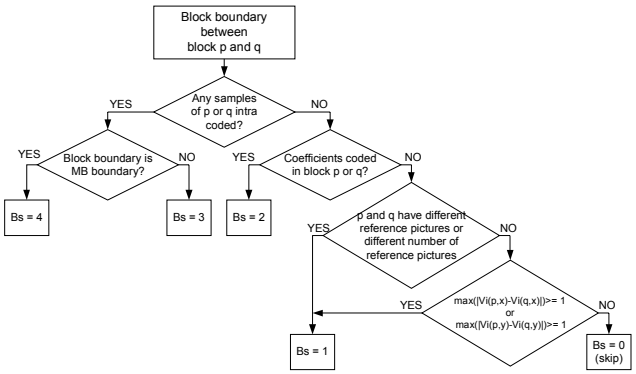

Fig. 4 Assignment of Boundary Strength (Bs)

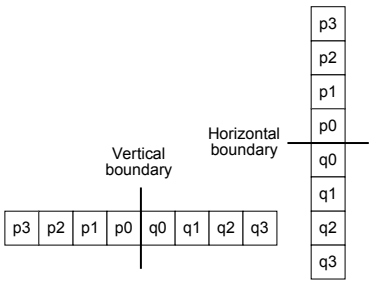

Fig. 5 Pixels across Vertical or Horizontal Boundaries

The deblocking filter is conditional. The outputs depend on the boundary strength and the gradient of image samples across the boundary. The boundary strength parameter "Bs" is assigned as described in Fig. 4. If one of the neighboring blocks is intra coded and the boundary is a MB boundary, the strongest filter processing is applied (Bs=4). If one of the neighboring blocks is intra coded but the boundary is not a MB boundary, a relatively strong filter processing is applied (Bs=3). If neither of the blocks is intra coded and at least one of them contains coded coefficients, a medium strength is assigned (Bs=2). Otherwise, if the blocks have different reference frames or different numbers of reference frames or different motion vector values, Bs=1. When none of the previous conditions is satisfied, the filter processing is not evoked (Bs=0). The boundary strength of any chroma component is the same with that of luma component on the same edge.

Fig. 5 shows four pixels on either side of a vertical or horizontal boundary in adjacent blocks p (p0, p1, p2, p3) and q (q0, q1, q2, q3). Each filtering operation affects up to three pixels on either side of the boundary, which is dependent on Bs, quantization parameters (QP) of the neighboring blocks and the

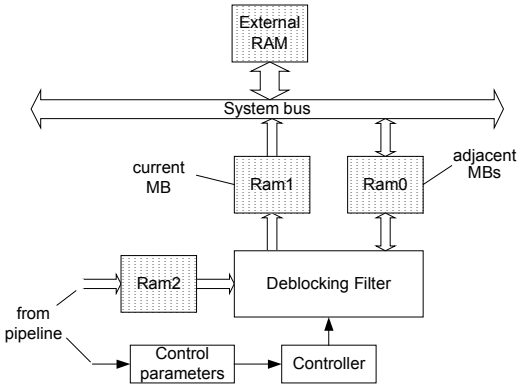

Fig. 6 Deblocking Filter Architecture

two control parameters "FilterOffsetA" and "FilterOffsetB". A group of samples from the set {p2, p1, p0, q0, q1, q2} are filtered only if the following conditions

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

are all satisfied. Here α and β are thresholds that depend on QP, FilterOffsetA and FilterOffsetB.

(a) If $Bs \in \{1,2,3\}$, a 4-tap linear filter is applied with inputs p1, p0, q0 and q1, producing filtered outputs P0 and Q0. For luma component, if $|p2 - p0| < \beta$, a 4-tap linear filter is applied with inputs p2, p1, p0 and q0, producing filtered output P1. Similarly, if $|q2 - q0| < \beta$, a 4-tap linear filter is applied with inputs q2, q1, q0 and p0, producing filtered output Q1.

(b) If $Bs = 4$, a 3-tap filter, a 4-tap filter or a 5-tap filter may be applied to produce P0, P1, P2, Q0, Q1 and Q2, depending on the thresholds α , β and actual values of the eight pixels.

For more information about the conditional filter, please refer to [1].

3. IMPLEMENTED ARCHITECTURE

The implemented architecture has been designed as a module that can be integrated into a pipeline-based decoder or encoder. The input pixel data of the deblocking filter come from two modules. One is an on-chip SRAM ("Ram2" in Fig. 6) that contains unprocessed pixel data of the current MB (16 luma 4x4 blocks and 8 chroma 4x4 blocks in 4:2:0 mode) from the prior modules in the pipeline, namely, inverse transformation, motion compensation and intra prediction. The other is on-chip SRAM "Ram0" that contains the adjacent pixel data (8 luma 4x4 blocks and 8 chroma 4x4 blocks) of the top and the left MBs of the current MB. We should load "Ram0" with needed pixel data from "external RAM" via system bus in advance. The processing results are sent into two on-chip SRAMs. One is "Ram1" for the pixel data of current MB. The other is "Ram0" for the adjacent pixel data of the top and the left MBs. After deblocking filtering, processed data in "Ram0" and "Ram1" have to be stored back to "external RAM".

3.1 Organization of SRAMs

Fig. 7 shows the organization of "Ram0", "Ram1" and "Ram2". The three SRAMs have the same bit width, 32 bits for four pixels. The shadowy blocks in Fig. 7 are the adjacent pixel data in the top

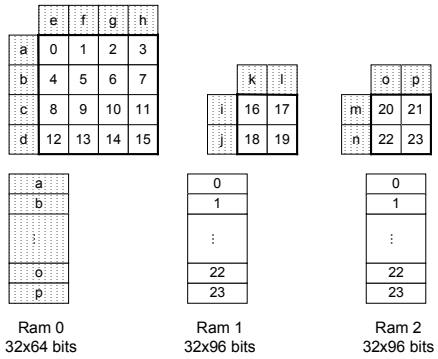

Fig. 7 Organization of On-chip SRAM modules

or the left MBs of the current MB. Moreover, the rest blocks (from 0 to 23) belong to the current MB.

3.2 Processing Order

In H.264/AVC, the processing order of deblocking filter is that, horizontal filter across vertical boundaries is done firstly; vertical filter across horizontal boundaries is evoked sequentially. According to this rule, there are two kinds of processing orders previously, which are shown in Fig. 8 (a) and (b).

The processing order, described in Fig. 8 (a), is the basic one obeying the rule of H.264/AVC deblocking filter [1] [7]. Nevertheless, this filtering order does not make use of data dependence between neighboring 4x4 blocks. For example, when processing the boundary between “a” and “0” (referring to Fig. 7), we should load them from on-chip SRAMs firstly. The outputs are “ a' ” and “ 0' ” respectively. If we filter the boundary between “b” and “4” in succession according to the basic filtering order, the data of “ 0' ” have to be stored back to SRAM. When processing the boundary between “0” and “1”, we have to load “ 0' ” from SRAM secondly, and store it back to SRAM after filter processing for the second time. This is only the case of horizontal filtering across vertical edges. Including the case of vertical filtering across horizontal edges, data of 4x4 block “0” have to be loaded out from and stored back to the on-chip SRAM four times respectively in total. Obviously, the basic processing order has a high requirement of on-chip SRAM bandwidth.

Fig. 8 (b) shows an “advanced filtering order”, which makes use of one-dimensional data dependence [6]. That is, after filtering between “a” and “0”, the output “ 0' ” will not be stored back to SRAM, but be sent to a buffer, as one input for the next filtering operation between “0” and “1”. In this way, any block will be loaded and stored only once in horizontal filtering, as such in vertical filtering. Half access requirements are avoided. Nevertheless, this isn’t the most efficient processing order because only 1-D data dependence is utilized.

Further more, we propose a more efficient 2-D processing order, which is described in Fig. 9. According to this order, horizontal filtering and vertical filtering are performed alternately. For example, “a” and “0” is filtered firstly. One result “ a' ” is stored back to “Ram0”, the other result “ 0' ” is buffered and is horizontally filtered with “1” in succession. After the last filtering, the output “ 1' ” is buffered (replace “ 0' ” in the buffer and to be filtered with “2” in the future), and the output “ 0" ” is sent to the matrix transposer. Then, the transposed “ 0" ” is vertically filtered

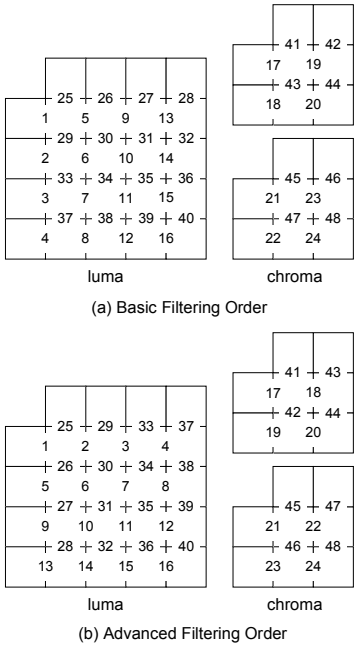

Fig. 8 Previous Processing Orders

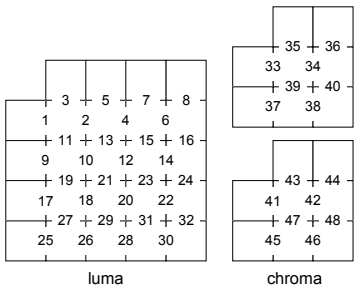

Fig. 9 Proposed 2-D Processing Order

with “e” (referring to Fig. 7). The rest may be deduced similarly. The data dependence between neighboring blocks in both horizontal direction and vertical direction is utilized fully. Although extra buffers and matrix transposers are needed, the requirement of on-chip SRAM bandwidth is declined and the throughput is enhanced evidently, which is very important for the real-time deblocking of HDTV (1280x720, 60fps) video.

3.3 Implemented Architecture

Fig. 10 shows block diagram of the 2-D deblocking filter. In order to simplify our design, “Ram0” is assigned to a dual-port (two read/write ports) SRAM, “Ram1” and “Ram2” are assigned to two-port SRAMs (one read port and one write port). The 16 4x4 blocks, in the adjacent MBs of the current MB, are loaded into “Ram0” in advance. Data of unprocessed 4x4 blocks in the current MB are ready in “Ram2” from the pipeline. Filtered 4x4 blocks are saved in “Ram1”, and to be stored back to “external RAM” at the end of the filter processing. “MT” is a matrix transposer, as shown in Fig. 11. Each cell of “MT” is an 8-bit register. “Buf” is a 32x4-bit buffer. The controller arranges the processing order of filtering according to the description in Fig. 9.

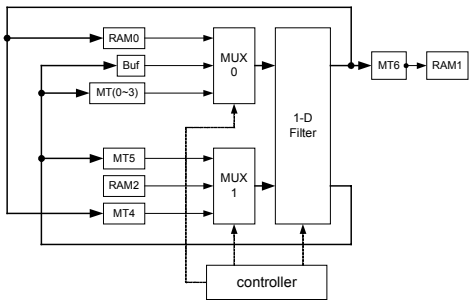

Fig. 10 Block Diagram of the 2-D Deblocking Filter

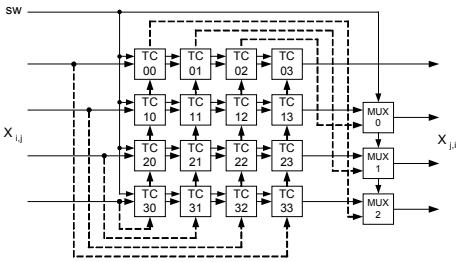

Fig. 11 Architecture of MT

Before Deblocking Processing (QP=37, PSNR=30.978)

After Deblocking Processing (QP=37, PSNR=31.375)

Fig. 12 Deblocking Filter's Effects

For convenience, it is assumed that the system data bus is 32-bit. The loading procedure of "Ram0" costs 4x16=64 cycles. The processing of conditional filtering, including luma and chroma components, costs 4x48=192 cycles. It costs 4x(16+24)=160 cycles to store "Ram0" and "Ram1" back to "external RAM". In our design, about 30 extra cycles are needed to load the coding information and to initialize the filter. To sum up, the number of total cycles for deblocking each MB is 446.

4. SIMULATION RESULTS

We described the design mentioned above in Verilog HDL at RTL level, which is synthesizable. According to JVT verification model [7], a C-program model of deblocking filter was also developed to generate input simulation vectors for VCS digital simulator. Tested with eight HDTV (1280x720, 60fps) bitstreams (100 frames per bitstream), our Verilog code is functionally identical with the deblocking filter of JVT verification model. Fig. 12 shows effects of our deblocking filter on the first frame (Intra) of "Foreman" sequence. Subjective views and PSNR values are all improved obviously.

Table 1 Comparison of Synthesized Results

	Yuwen Huang's Architecture	Our Architecture
Technology	0.25μm	0.25μm
Working Frequency	100MHz	100MHz
Gate Count (Without SRAM)	20.66K	24K
Cycles/MB	614	446
Capacity	1280x720 45.2fps	1280x720 62.3fps
Bus Bandwidth (Mbytes/s) 1280x720 30Hz 1280x720 60Hz	 138.24 (not supported)	 96.768 193.536

The validated Verilog code was synthesized using 0.25μm CMOS cells library by Synopsys Design Compiler. The circuit costs about 24k logic gates (not including a 32x64 SRAM and two 32x96 SRAMs) when the working frequency is set to 100MHz. Table 1 shows the comparison of synthesized results between Yuwen Huang's design in [6] and ours. Our architecture costs 446 cycles to perform deblocking filter for each MB. It is sufficient to realize the real-time deblocking filter of a HDTV (1280x720, 60fps) H.264/AVC bitstream.

5. CONCLUSIONS

In this paper, we implemented a VLSI architecture of deblocking filter for H.264/AVC. Firstly, we explained the algorithm of H.264 /AVC deblocking filter. Then we proposed an efficient processing order for the deblocking filter. Making good use of the data dependence between neighboring 4x4 blocks, in both horizontal direction and vertical direction, the processing order reduces the requirement of on-chip SRAM bandwidth and increases the throughput of the filter processing. According to the order, the VLSI architecture is designed. The synthesized results indicate that our design can support real-time deblocking filter of HDTV (1280x720, 60fps) H.264/AVC video. The architecture is valuable for the hardware design of H.264/AVC CODEC.

6. REFERENCES

[1] Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC), Geneva, Switzerland, May. 2003.

[2] ITU-T, Draft ITU-T Recommendation H.263, "Video Coding for Low Bit Rate Communication," 1997.

[3] ISO/IEC IS 13818, "General Coding of Moving Picture and Associated Audio Information," 1994.

[4] ISO/IEC FCD 14496, "Information technology – Coding of audio-visual objects – Part 2: Visual," July 2001.

[5] M.T. Orchard and G. J. Sullivan, "Overlapped Block Motion Compensation: An Estimation-Theoretic Approach," IEEE Trans. Image Processing, pp. 693-699, September 1994.

[6] Yuwen Huang, Towei Chen, "Architecture Design for Deblocking Filter in H.264/AVC," in Proceedings of ICME, pp. 693-696, Baltimore, Maryland, USA, July 6-9, 2003.

[7] JVT software JM7.3, August 2003.