ARCHITECTURE DESIGN FOR DEBLOCKING FILTER IN H.264/JVT/AVC

Yu-Wen Huang, To-Wei Chen, Bing-Yu Hsieh, Tu-Chih Wang, Te-Hao Chang, and Liang-Gee Chen*

DSP/IC Design Lab., Graduate Institute of Electronics Engineering and
Department of Electrical Engineering, National Taiwan University
{yuwen, tchen, bingyu, eric, thchang, lgchen}@video.ee.ntu.edu.tw

ABSTRACT

This paper presents an efficient VLSI architecture for the deblocking filter in H.264/JVT/AVC. We use an array of 8x4 8-bit shift registers with reconfigurable data path to support both horizontal filtering and vertical filtering on the same circuit (a parallel-in parallel-out reconfigurable FIR filter). Two SRAM modules are carefully organized not only for the storage of current macroblock data and adjacent block data but also for the efficient access of pixels in different blocks. Simulation results show that under 0.25 μm technology, the synthesized logic gate count is only 19.1 K (not including a 96x32 SRAM and a 64x32 SRAM) when the maximum frequency is 100 MHz. Our architecture design can easily support real-time deblocking of 720p (1280x720) 30Hz video. It is valuable for platform-based design of H.264 codec.

1. INTRODUCTION

Experts from ISO/IEC MPEG-4 Advanced Video Coding (AVC) and ITU-T H.264 cooperate together as the Joint Video Team (JVT) to develop the emerging standard [1]. The new standard significantly outperforms previous ones in bit-rate reduction. The functional blocks of H.264, as well as their features, are shown in Fig. 1. Unlike the overlapped motion compensation (OBMC) [2] in H.263 [3] and MPEG-4 [4], H.264 adopts the deblocking filter to eliminate blocking artifacts and to achieve much better subjective views. Since the deblocking filter is located in the DPCM loop, it is required at both the encoder side and the decoder side. The deblocking filter is much more complex than common low-pass FIR filters that introduce blurring effects on real edges. The concept of deblocking is to first decide whether the discontinuity between block boundaries is resulted from quantization, different motion vectors, or real edges. Then, different types of filters are selected according to the decision in order to maintain the sharpness of real edges and to smooth the unpleasant block boundaries simultaneously. The number of taps, set of filter coefficients, clipping functions, and threshold values, are all adaptive with different coding modes.

At the decoder side, deblocking filter contributes to a considerable amount of computation. At the encoder side, motion estimation becomes the processing bottleneck. However, pure software implementation of deblocking filter still requires a large amount of bus bandwidth for platform-based design. For example, CIF (352x288) 30Hz video needs 36 Mbytes/sec, and 720x480 30Hz video needs 124 Mbytes/sec. Therefore, we proposed a hardware accelerator for deblocking to decrease the processing cycles and the heavy burden of system bus. The rest of this paper is organized as follows. In Section 2, we will review the algorithm of deblocking in H.264. In Section 3, we will describe our architecture design. Simulation results will be shown in Section 4. Finally, Section 5 gives a conclusion.

*The author thanks SiS Education Foundation for financial support.

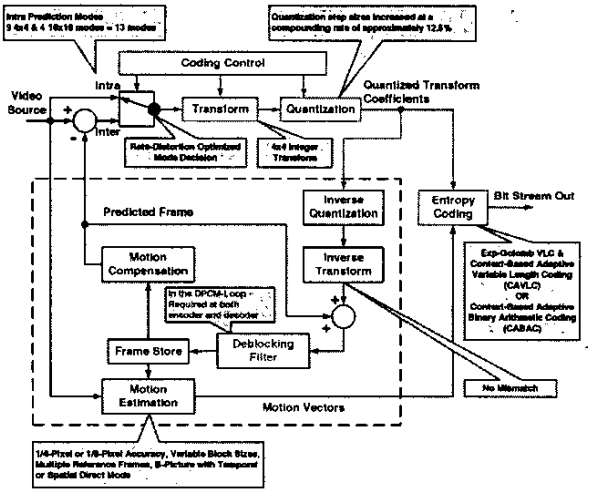


Fig. 1. Functional blocks and features of H.264.

2. ALGORITHM

In H.264, the transformation is based on 4x4-blocks, and the smallest block size for motion compensation is also 4x4. Therefore, deblocking process checks the boundaries between 4x4 blocks. Deblocking is done macroblock (MB) by macroblock in raster scan order. In each macroblock, horizontal filtering across vertical edges is first executed, and then vertical filtering across horizontal edges is applied. The sequential order of edges is shown in Fig. 2. For each boundary between neighboring 4x4-luma blocks, a boundary strength (Bs) is assigned as shown in Fig. 3. If one of the neighboring blocks is intra-coded, a relatively strong filtering (Bs=3) is applied. If the block boundary is also macroblock boundary and one of the blocks is intra-coded, an even stronger filtering procedure is applied (Bs=4). If neither of the blocks are intra-coded and at least one of them contains non-zero transform quantized residues, medium filtering strength (Bs=2) is used. If none of the previous conditions are satisfied, Bs=1 when the ref-

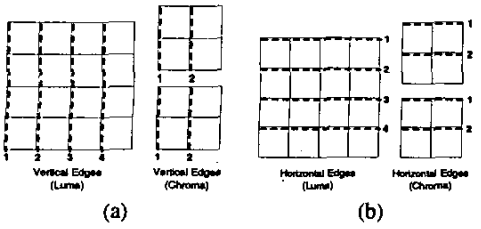


Fig. 2. Sequential order of vertical edges and horizontal edges in a macroblock. Each square stands for a 4x4-block.

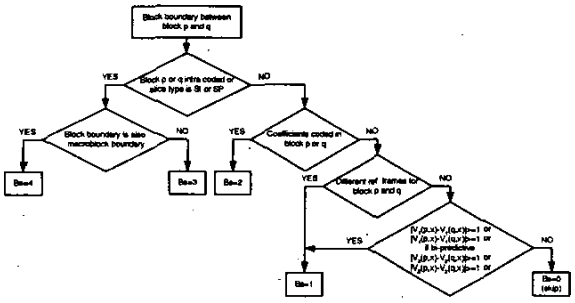


Fig. 3. Flowchart for determining the boundary strength.

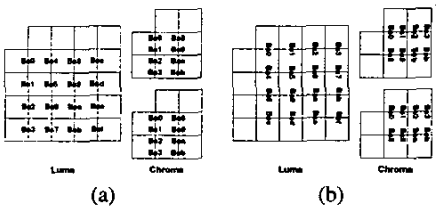


Fig. 4. Chroma boundary strengths.

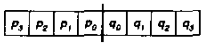


Fig. 5. Convention for describing samples across a boundary between two 4x4-blocks (in horizontal or vertical direction).

Fig. 6. Subjective views and PSNR values without/with deblocking for the first frame of Foreman.

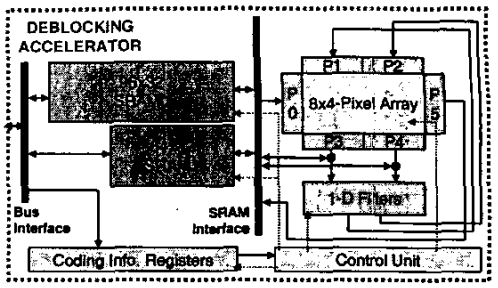


Fig. 7. Architecture design for the deblocking filter.

erence frames of two blocks are different or when the reference frames are the same but any component of the two motion vectors has difference more than one pixel sample. Otherwise filtering is skipped (Bs=0). The chroma boundary strengths are the same as corresponding luma boundary strengths, as shown in Fig. 4. In the following description, the set of eight samples across a boundary between two 4x4-blocks (in horizontal or vertical direction) is denoted as shown in Fig. 5 with the actual boundary lying between p_0 and q_0. Sets of samples across this edge are only filtered if the following conditions

$$Bs \neq 0 \quad |p_0 - q_0| < \alpha \quad |p_1 - p_0| < \beta \quad |q_1 - q_0| < \beta$$

are all true. Note that α and β are QP (quantization parameter) dependent thresholds. After Bs is determined, two types of filtering are specified. In the default case (0<Bs<4), p_0 and q_0 are filtered with QP dependent clipping functions. For luma samples, p_1 and q_1 are further conditionally filtered. The number of taps is four. In the other case (Bs=4), a 3-tap filter, a 4-tap filter, or a 5-tap filter is applied on p_2, p_1, p_0, q_0, q_1 and q_2, which depends on the local activity of the luma or chroma eight samples and QP dependent thresholds. Let us skip the details of the coefficients of filters. Interested readers can refer to [1] and [5]. Figure 6 shows the effect of the deblocking filter.

3. ARCHITECTURE

Figure 7 shows the proposed architecture design for the deblocking filter. The solid arrows denote data path, and the dotted arrows denote control signals. Before deblocking a macroblock for platform-based design, we have to load the macroblock data (16 luma 4x4-blocks, 8 chroma 4x4-blocks) and adjacent block data (8 luma 4x4-blocks, 4 chroma 4x4-blocks) from external RAM via system bus to the on-chip SRAM. A parallel-in parallel-out (eight pixels in, eight pixels out) 1-D reconfigurable FIR filter is directly implemented as stated in the previous section. In order to support the parallel filter with high utilization, two SRAM modules are carefully organized. As shown in Fig. 8, the bit width of each SRAM is 32 (4 pixels). We classify 4x4-blocks into different columns (c0-c10) and store 4x1 pixels as one 32-bit word in SRAM. Then, we put adjacent columns of block data in different SRAM modules so that we can access any eight pixels across different columns of blocks. In this way, horizontal filtering across vertical edges becomes very easy. However, vertical filtering across horizontal edges are not that straightforward. Our solutions will be stated as follows.

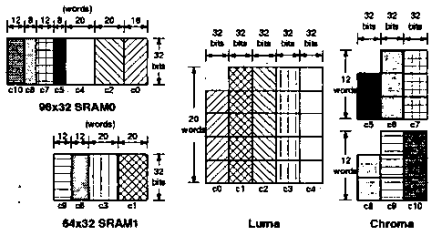


Fig. 8. Organization of on-chip single port SRAM modules.

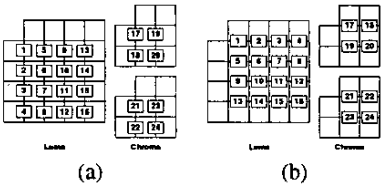


Fig. 9. Processing order of boundaries; (a) horizontal filtering across vertical edges; (b) vertical filtering across horizontal edges.

3.1. Basic Architecture with 2 Single Port SRAM's

In this subsection, we assume each SRAM module has only one read/write port. The processing order of block boundaries for both directions is shown in Fig. 9. Our basic idea is to buffer the 8x4 unfiltered/filtered pixels of two adjacent 4x4-blocks in an 8x4 pixel array with reconfigurable data path in order to support both horizontal and vertical filtering on the same 1-D filters.

The data path for horizontal filtering is shown in Fig. 10(a). Note that solid arrows denote enabled path while dotted arrows denote disabled path, and each square stands for an 8-bit register. For this architecture, it takes 8 clock cycles to process each pair of 4x4-blocks. In the first 4 cycles, 8x1 pixels are inputted from two SRAM modules into 1-D filters. Meanwhile, the filtered pixels are fed to Port1 and Port2 of the pixel array with downward path. In the later 4 cycles, filtered 8x4 pixels are already saved in the array, and we store them back to the two SRAM's.

The data path for vertical filtering is divided into a load/store phase and a filtering phase, as shown in Fig. 10(b) and (c), respectively. First, we assume the filtered 8x4 pixels of previous boundary are already buffered in the array. In the load/store phase, the path of pixel array is rightward. It takes 8 cycles to load the 4x8 pixels of a 4x4-block pair belonging to one column, and also to store the 4x8 previously filtered pixels of another 4x4-block pair belonging to the previous column. The load and store can be executed at the same time without conflict because of the interleaved memory organization. In the filtering phase, the path of pixel array becomes downward. It is clearly that 4 cycles are required to filter the 4x8 pixels and to buffer the filtered results in the array.

In sum, if the bit width of system bus is typically 32, we have to spend 160 cycles to load unfiltered pixels from external memory to on-chip SRAM's, 8x24=192 cycles to filter in horizontal direction,(8+12x16)+(8+12x4)x2=312 cycles to filter in vertical direction, and 160 cycles to write filtered pixels from on-chip SRAM's to external memory. Some extra cycles are also required to load the coding information before deblocking. In our implementation,

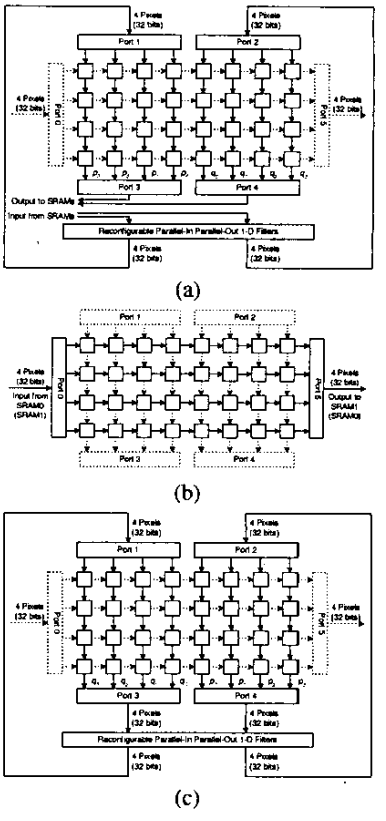


Fig. 10. Data path; (a) horizontal filtering across vertical edges; (b) vertical filtering across horizontal edges (load/store phase); (c) vertical filtering across horizontal edges (filtering phase).

54 cycles are required. The number of total cycles required for deblocking a macroblock is 878.

3.2. Advanced Architecture with 1 Dual Port SRAM

We can pack the pixels, which were originally placed in two single port SRAM's as shown in Fig. 8, together into one dual port SRAM as illustrated in Fig. 11, without affecting the data flow. Because a dual port SRAM has two separate read/write ports, it can perform two read operations at the same cycle, as well as two write operations, or one read and one write at the same cycle. It is worthwhile to use one dual port SRAM instead of two single port SRAM's for we can save one power ring on the physical layout.

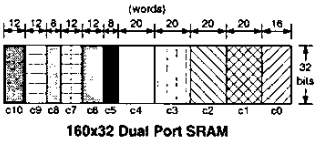
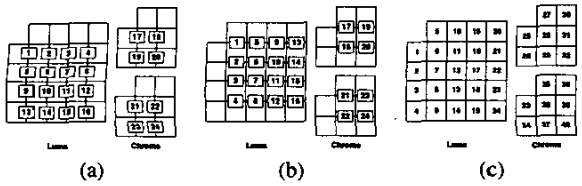


Fig. 11. Organization of on-chip dual port SRAM.



(a) (b) (c)

Fig. 12. Advanced processing order of boundaries; (a) horizontal filtering; (b) vertical filtering; (c) block index.

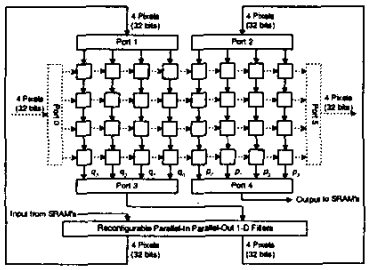


Fig. 13. Advanced data path for horizontal filtering.

Next, we modify the processing order of boundaries as shown in Fig. 12(a)(b) without affecting the data dependency in Fig. 2. The block index is shown in Fig. 12(c). The main idea is stated as follows. For example, for horizontal filtering, after boundary 1 is processed, we only have to write block 1 from array to SRAM. As for block 6, we can directly send it back to the filters with block 11 from SRAM to keep on processing boundary 2. At this moment, in order to support the new processing order, the data path also has to be reconfigured as Fig. 13. In this way, the number of cycles for horizontal filtering is decreased to 152. The data flow and data path of vertical filtering can be easily derived with the help of Fig. 12(b). The number of cycles for vertical filtering is reduced to 288, and the number of total cycles becomes 814.

3.3. Parallel Architecture with 2 Two Port SRAM's

If we replace each of the two single port SRAM's by a two port SRAM (one read port and one write port), for horizontal filtering, we do not need the 8x4 pixel array. We can directly get 8x1 pixels from two SRAM's, filter 8x1 pixels, and write them back to SRAM's at the same time. However, for vertical filtering, the 8x4 pixel array is still necessary because the vertical filtered 8x1 pixels belong to 8 separate words in one SRAM. That is, the data path for vertical filtering is still the same as Fig. 10(b) and (c). Therefore, it only requires 4x24=96 cycles for horizontal filtering, and the number of total cycles is reduced to 782. If we can have 2 two port SRAM's and dual 8x4 pixel arrays in parallel, the vertical filtering can be further accelerated. In this way, the number of cycles required for vertical filtering becomes 12x24/2=144, and the number of total cycles becomes 614.

Table 1. Comparison of architectures synthesized at 100 MHz.

Architectures	Gate Count	Cycles/MB	Capability
Basic + single port SRAM's	18.91K	878	1280x720 31.6fps
Advanced + dual port SRAM	18.91K	814	1280x720 34.1fps
Basic + two port SRAM's	18.91K	782	1280x720 35.5fps
Dual arrays + two port SRAM's	20.66K	614	1280x720 45.2fps

Filters: 6.67K gates; Array: 1.76K gates; Coding Info. Registers: 5.32K gates. RAM is not included in Gate Count.

Table 2. Comparison of bus bandwidth.

Specifications	Software (RISC) (Mbytes/sec)	Our Architecture (Mbytes/sec)
CIF (352x240) 30Hz	36.50	15.21
NTSC (720x480) 30Hz	124.42	51.84
720p (1280x720) 30Hz	331.78	138.24

4. SIMULATION RESULTS

We described our architecture by Verilog HDL and synthesized the circuit using 0.25 μm Artisan CMOS cell library by Synopsys Design Analyzer with critical path constraint set to 10 ns (100 MHz). The results are shown in Table 1. The processing capability of our architecture can easily support real-time deblocking of 720p 30Hz video, and the synthesized gate count is very small. Table 2 shows the comparison of system bus bandwidth for platform-based design. Our architecture can save more than 50% of the bandwidth compared to software implementation of the deblocking filter (processed by RISC). According to the simulation results, our design is a good candidate of deblocking filter for the platform-based design of H.264 coding systems.

5. CONCLUSION

In this paper, we contributed a hardware deblocking accelerator for H.264/JVT/AVC. The major idea is to use interleaved memory organization and an 8x4 pixel array with reconfigurable data path to support one parallel-in parallel-out reconfigurable 1-D filter. Simulation results show that the processing capability of the proposed architecture is very high (real-time deblocking of 1280x720 30Hz video operating at 66-94 MHz), and the area is very small (around 19K logic gates + 96x32 SRAM + 64x32 SRAM). It is very suitable to be integrated into H.264 coding systems.

6. REFERENCES

[1] *Committee Draft of Joint Video Specification (ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC)*, July, 2002.

[2] M.T. Orchard and G.J Suillivan, "Overlapped block motion compensation: an estimation-theoretic approach," *IEEE Trans. Image Processing*, pp. 693-699, Sep. 1994.

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

[4] *Information Technology - Coding of Audio-Visual Objects - Part 2: Visual, ISO/IEC 14496-2*, 1999.

[5] *Joint Video Team (JVT) software JM6.1d*, March, 2003.