Journal of Zhejiang University SCIENCE A
ISSN 1673-565X (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Parallel processing architecture of H.264 adaptive deblocking filters*

Hu WEI†, Tao LIN, Zheng-hui LIN

(*Department of Electronic Engineering, Shanghai Jiao Tong University, Shanghai 200240, China*)

†E-mail: microtiger@sjtu.org

Received July 1, 2008; Revision accepted Oct. 26, 2008; Crosschecked May 27, 2009

Abstract: In H.264, computational complexity and memory access of deblocking filters are variable, dependent on video contents. This paper proposes a VLSI architecture of deblocking filters with adaptive dynamic power, which avoids redundant computations and memory accesses by precluding the blocks that can be skipped. The vertical and horizontal edges are simultaneously processed in an advanced scan order to speed up the decoder. As a result, dynamic power of the proposed architecture can be reduced adaptively (up to about 89%) for different videos, and the off-chip memory access is improved when compared to previous designs. Moreover, the processing capability of the proposed architecture is in particular appropriate for real-time deblocking of high-definition television (HDTV, 1920×1080 pixels/frame, 60 frames/s video signals) video operation at 62 MHz. Using the proposed architecture, power can be reduced by up to about 89% and processing time by from 25% to 81% compared with previous designs.

Key words: Deblocking filter, Adaptive dynamic power, Parallel processing, Pipeline, H.264
doi:10.1631/jzus.A0820502 **Document code:** A **CLC number:** TN919.8

INTRODUCTION

The block-based video coding scheme has been widely used in various coding standards, such as MPEG-1, MPEG-2, MPEG-4 and H.264. Computation regularity and its simple architecture are outstanding features of such a block-based coding scheme, but block based processing makes the blocking artifact. As to the reduction of blocking artifacts produced by block-based video coding, deblocking filters are used; for example, H.264/AVC adopts an adaptive in-loop deblocking filter (Sullivan *et al*., 2004) that reduces the bit rates typically by 10%~15% at the same visual fidelity. The only problem associated with deblocking filters is their high computational complexity. The operation of these filters is the most complex part of the decoders, consuming one-third of the computational complexity of the H.264/AVC decoder (Lou *et al.*, 2007). Thus, fast computation of deblocking filters is necessary for real time processing of the H.264/AVC decoder.

Previous research of fast deblocking filters focuses on two fields: pure software implementations and hardware implementations. Filter complexity is mainly due to high adaptivity conducive to heavy conditional computations that are executed in the inner loop of the algorithm, occupying about 30% to 90% of the overall computation time (Lou *et al.*, 2007). Another reason for the high complexity is the small block size employed for residual coding in the H.264/AVC. Each pixel of the 4×4 block has to be loaded from the memory for conditional checking, which is a big challenge for software implementations during parallel processing under DSP (digital signal processor) or SIMD (single instruction multiple data) computational architecture (Warrington *et al.*, 2006; Arbelo *et al.*, 2007; Lou *et al.*, 2007). When compared to software implementations, a hardware implementation of in-loop deblocking filters is usually preferred. Cheng and Chang (2005) proposed an efficient one-dimensional processing order to reuse intermediate data to reduce memory accesses. In (Huang *et al*., 2003; 2005; Li *et al*., 2005; Chen and Chen, 2005; Liu *et al*., 2007; Min and Chong, 2007; Zhao and

* Project (No. NSS' USA5978) supported by the National Science Foundation of the United States under the East Asia Pacific Program

Jiang, 2007; Hao and Radetzki, 2008; Xu and Choy, 2008), the advanced two-dimensional processing orders were proposed to reduce operation cycles and memory accesses. In (Chen and Chen, 2006; 2007; Zhao and Lu, 2007), two edge-filtering units were adopted to parallelize the deblocking process, accelerating the operation of deblocking filters and reducing memory accesses. Vladimir and Ivan (2007) made a tradeoff between area and time to reduce the chip area.

Deblocking filters described in the H.264/AVC standard are highly adaptive. Several parameters and thresholds, as well as the content of the picture itself, control the boundary strength (BS) of the filtering process. With conditional checking, the filtering operations for an edge can be skipped, and therefore the efficient edge filtering order for a macroblock (MB) is variable. This challenges the low power implementations of the deblocking filter, especially in the pipelined architectures. All previous designs filter the block edges in some fixed orders to simplify the architecture, which loads and stores the block data for a skipped edge in the same way as for a non-skipped edge and let them transfer through the whole data path of the deblocking pipeline. Thus, some redundant power and redundant memory accesses occur during the deblocking process. In this paper, we propose a parallel-pipelined architecture for H.264 deblocking filters which have adaptive power and memory accesses. Using the proposed architecture, power can be reduced by up to about 89% and processing time by from 25% to 81% compared with previous designs.

PARALLEL PROCESSING IN H.264 DEBLOCKING FILTERS

Introduction of H.264 deblocking filters

In this work we describe a deblocking algorithm for the 4:2:0 video format in which an MB comprises one 16×16 luminance block and two 8×8 chrominance blocks (Cr and Cb). In H.264/AVC, the deblocking filter is applied on each 4×4 luminance and chrominance block edge on an MB basis, as shown in Fig.1. The horizontal or vertical edges to be filtered for an MB are denoted with bold or dashed lines in Fig.1a, in which dashed edges can be skipped if inside an 8×8 transform block for H.264 FREXt (Sullivan *et al*., 2004). Edge filtering is based on the line of pixels (LOP), which is made up of 8 pixels (p_3, p_2, p_1, p_0, q_0, q_1, q_2, q_3) across the edge. In an MB, horizontal LOPs are filtered first from left to right, and then vertical LOPs are filtered from top to bottom. For a 4×4 block, each pixel must be loaded and stored four times at most, which endues the deblocking filter with a lot of memory accesses. As seen in Fig.1b, three edges of blocks A or B and two edges of blocks C or D can be filtered successively. The filtered block data can thus be reused immediately as input for filtering the next edge of the block, making it possible to reduce memory accesses and deblocking cycles.

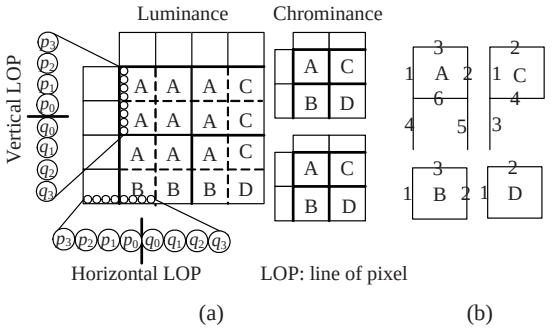


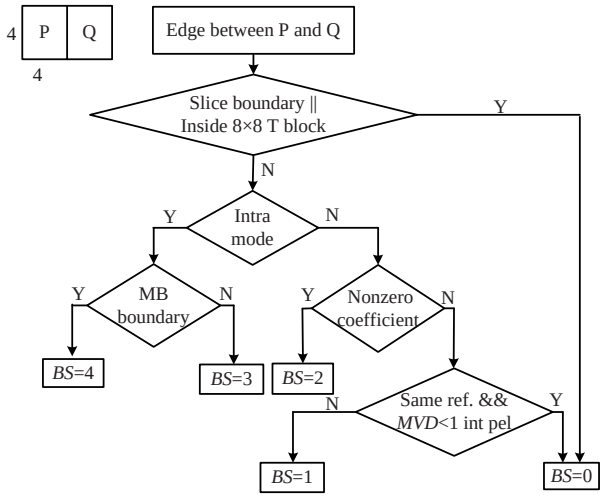
Fig.1 (a) The edges to be filtered in an MB; (b) The order of edge filtering for a block

Bold or dashed lines: the horizontal or vertical edges to be filtered for an MB. Dashed edges can be skipped if inside an 8×8 transform block for H.264 FREXt

The deblocking filter algorithm of H.264 is highly adaptive at three different levels: slice level, block edge level, and sample level. At the slice level, the threshold parameters are defined for the filtering operation. At the block edge level, the boundary strength of the filtering operation is made dependent on inter/intra predictions, motion vector difference, and coded residuals in the 4×4 blocks (Fig.2). At the sample level, it is decided whether to turn on the filter for a particular sample to avoid real edge distortion according to the conditions listed below:

$$|p_0-q_0|<\alpha,\ |p_1-p_0|<\beta,\ |q_1-q_0|<\beta,\ BS\neq 0,$$

where α and β are thresholds dependent on the quantization parameter QP.



T: transform
MVD: motion vector difference
Int pel: integer pixel

Fig.2 Decision flow of boundary strength (BS) for the edge between adjacent blocks P and Q

Parallel processing of vertical and horizontal edges

A LOP filter can filter a line of pixels in a cycle. To speed up the process of deblocking, parallel processing with more than one LOP filter is helpful. We use two LOP filters to process vertical and horizontal edges simultaneously. Input of the vertical LOP filter comes from output of the horizontal LOP filter. Compared to parallel processing of luma and chroma components, the two LOP filters in our architecture undertake the same workload for different video formats (such as 4:2:0, 4:2:2 and 4:4:4) but work faster for 4:2:0 format video.

When all the edges of an MB must be filtered, the edge processing order is designed as in Fig.3. In this order blocks A and B (Fig.1) can be immediately reused twice, and the deblocking filter may be implemented with a block-based pipeline because the edges of each block have the same delay. It means that the number of blocks in the MB determines the speed of deblocking. Because the left four luma blocks and four chroma blocks in Fig.3 have only one edge to be filtered, we forbid them from taking the whole pipeline data path to reduce power and buffer them in a local memory to reduce external memory accesses and cycles.

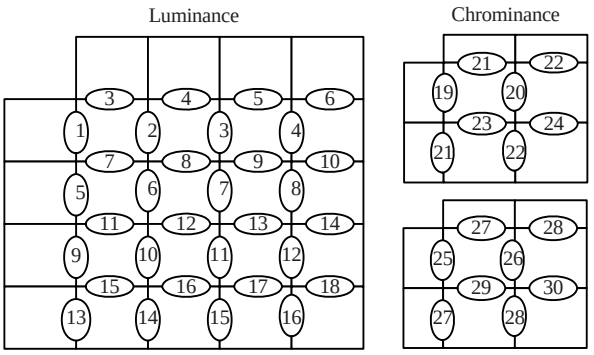
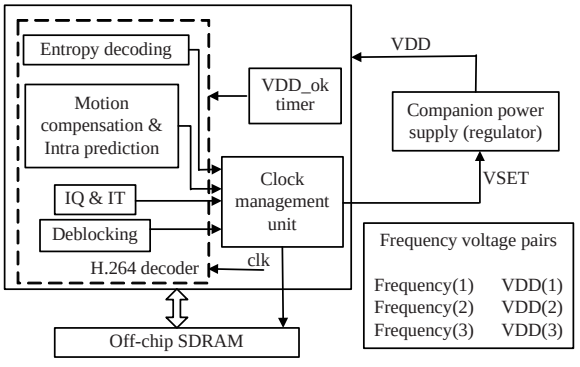


Fig.3 Parallel processing order of vertical and horizontal edges

SCHEME OF ADAPTIVE POWER CONTROL FOR DEBLOCKING FILTERS

H.264 decoder with adaptive dynamic power

Because computational complexity and memory access of H.264 decoders depend on video contents, the adaptive power technology can be used in the system architecture design for such decoders, in particular very useful for prolonging battery time of mobile products. The architecture of an H.264 decoder with adaptive power is shown in Fig.4. Each unit of the decoder works under the synchronous system clock, and the decoder works under the maximum frequency at a high load, while operates under a lower frequency at a low load. The chip voltage is adjusted with frequency for better power saving.



IQ: inverse quantization, IT: inverse transform
VDD: drain voltage, VSET: voltage set

Fig.4 Architecture of adaptive dynamic power H.264 decoder

Each unit of the decoder in Fig.4 generates power adjusting information to the clock management unit, which decides the system clock. The deblocking filter has, as we know, high computational complexities and memory accesses, which impact on the decision of system clock. In general, video decoder chips use external SDRAM or DDRAM as the frame buffers. The lowest frequency to run the decoder chip is limited by the off-chip memory access. Hence, speeding up the deblocking process and reducing the off-chip memory accesses for the deblocking process can reduce the power of the decoder.

In this study, we adopt two methods to reduce the external memory accesses of deblocking filters:

(1) Do not load the input data of deblocking filters from the off-chip memory; instead, read them from the reconstructed MB buffer (after prediction and inverse transformation) on chip.

(2) Store the rightmost 4 luma blocks and 4 chroma blocks of the MB into a local buffer after deblocking to reduce off-chip memory accesses.

In this way, upon switching on the deblocking filter, only 6 blocks (4 luma blocks and 4 half chroma blocks on topside of the MB) are increased for the off-chip memory access.

Scheme of adaptive power control for deblocking filters

The complexity of deblocking filters can be understood by the fact that each pixel of the 4×4 block has to be loaded from the memory for conditional checking and edge filtering. But the adaptivity of deblocking algorithms at the slice level and block edge level makes it possible to prejudge that an edge can be skipped before loading two blocks from memory. The criteria for prejudging a skipped edge are as follows:

$$BS=0 \text{ or } \alpha=0 \text{ or } \beta=0.$$

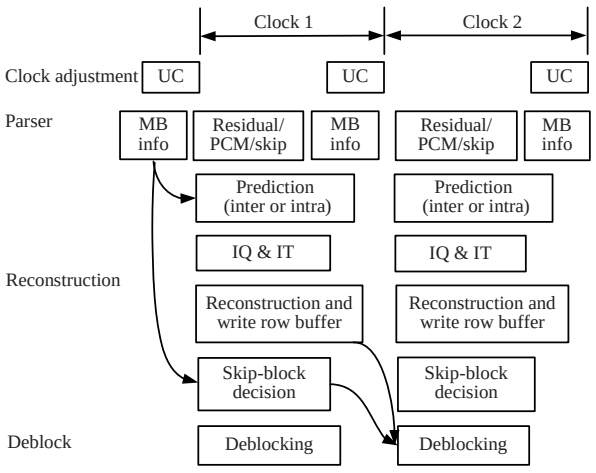
Table 1 shows the distribution of *BS* values for various video sequences under the H.264 main profile when the quantization parameter *QP*=26 dB. We can see that a lot of edges can be skipped without checking pixels (*BS*=0). In our experiments, the skipped edges in a P frame are often found being gathered in some blocks, and a lot of blocks have 4 skipped edges (called 'skip-block' herein).

Table 1 Percentage of different *BS* values for various sequences (QCIF, IPPPPPPPPPPP, *QP*=26 dB)

Sequence	Percentage (%)					Skip-block (%)
	BS=0	1	2	3	4	
Container	82.5	0.9	7.3	6.4	2.9	55.2
Foreman	47.3	32.7	13.9	4.1	2.0	18.4
Highway	65.0	17.6	7.9	6.3	3.1	37.5
News	78.0	5.2	11.2	3.8	1.8	47.5
Salesman	81.6	3.2	11.8	2.2	1.2	51.2
Silent	74.2	7.8	14.1	2.5	1.4	45.9

In our parallel processing architecture, reuse of intermediate block data brings the benefit of memory access reduction and deblocking cycle reduction, which is based on the 4×4 block pipeline with the advanced scan order. If skipping the edges randomly, the pipeline will be broken and the block data cannot be reused any more in the data path. Thus, we are inspired to skip the edges only if they are edges of a skip-block.

When skip-blocks in the MB are skipped, the necessary number of cycles for deblocking the MB is determined by the number of non-skipped blocks in the MB. We can therefore control the gated clock of the deblocking filter by pre-obtaining the information of skip- blocks before the deblocking process. Since the information for deciding skip-blocks in an MB is all encoded in the MB header syntax element, the process of skip-block decision can be run at the same time with MB reconstruction (Fig.5).

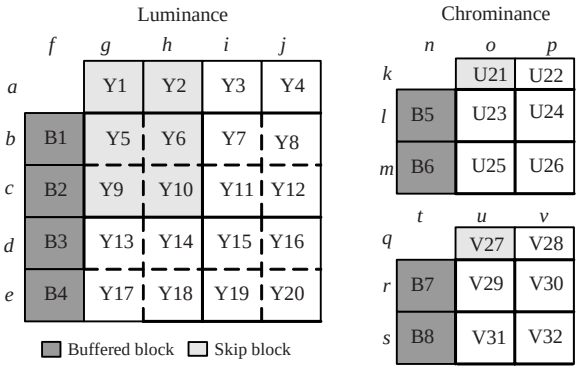


UC: update clock frequency, IQ: inverse quantization, IT: inverse transform

Fig.5 Macroblock level pipeline of H.264 decoding with adjusted system clock

In Fig.5 the deblocking process runs at an MB based pipeline. With the skip-block information, the off-chip memory access may also be pre-obtained to adjust the system clock and voltage.

We store the run number of skip-blocks into two arrays named 'skiprun' for luma and chroma respectively, which is in a raster scan order as the input order of the deblocking process (for an example of array skiprun, see Fig.6). Buffered blocks B1 to B8 are excluded from the array as they have been stored into deblocking filters.



Luma

Skiprun	Block
2	Y3
0	Y4
2	Y7
0	Y8
2	Y11
0	Y12
0	Y13
0	Y14
0	Y15
0	Y16
1	Y18
0	Y19
0	Y20

Chroma

Skiprun
2
0
2
0

Fig.6 An example of array skiprun

The adaptive power control process of deblocking filters is as follows:

(1) Compute *BS*, α and β values of each edge in the MB, and generate the array skiprun.

(2) Adjust power adaptively at the system level and the deblocking filter level.

At the system level, the off-chip memory accesses, being calculated with array skiprun, are used as the adjusted information for system clock and voltage (Fig.4). At the filter level, the gated clock of deblocking filters, being controlled according to array skiprun, is enabled only during the cycles of loading a non-skipped block into the deblocking filter. When the gated clock is disabled, move the skip-blocks from the reconstructed MB buffer to the off-chip memory bus directly, or buffer them into the local buffer to deblock the right MB.

IMPLEMENTATION

Macroblock level pipeline

The proposed parallel-pipelined deblocking filter with adaptive power has two levels of pipelines: MB level and block level. In the MB level, the preprocessing unit analyzes and generates the skiprun array, and then the parallel filter unit runs the deblocking process. The control unit controls the timing of the deblocking process and the gated clock. The block diagram of the proposed architecture is shown in Fig.7.

The preprocessing unit, in charge of the *BS* derivation, the thresholds derivation, and the skip-block analysis, generates the array skiprun, and counts the skip-blocks on topside of the MB to compute the increased off-chip memory accesses. The accesses are changed from 0 to 6 blocks (4 luma blocks and 4 chroma half blocks) for an MB.

The parallel filter units process the deblocking operations, including pixel checking and filter operation. The strong filter (*BS*=4), normal filter (*BS*=1, 2, 3) or bypass function is selected to finish the filter operations according to the *BS* value.

Parallel-pipelined filter unit

The parallel-pipelined filter unit processes the vertical and horizontal edges simultaneously, working under a block-based pipeline in the raster scan order. Luma and chroma components are processed in series. In this study, the memory bus width is 32 bits, and 4 luma or chroma samples in a row can be read/written in a cycle. In each block cycle (4 cycles in total) the filter unit loads a block and stores a block to external memory. Fig.8 shows the architecture of the proposed parallel-pipelined filter unit.

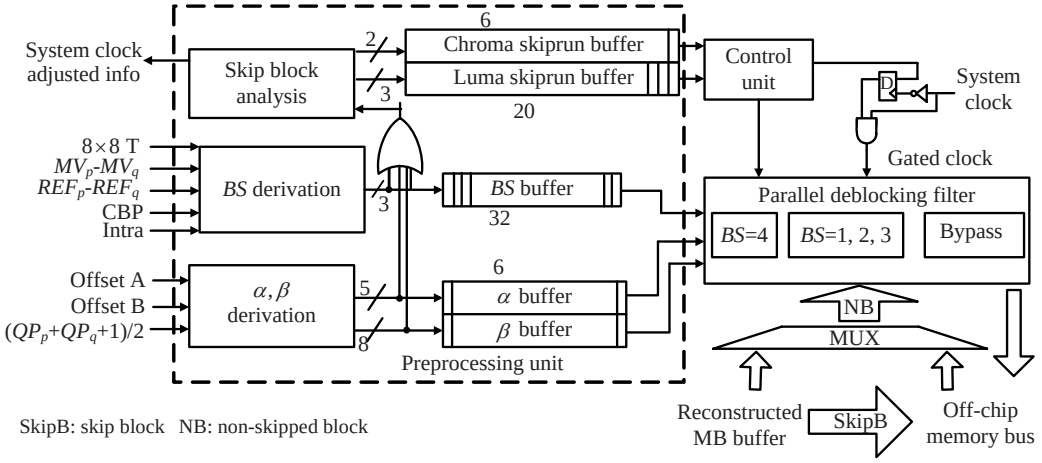


Fig.7 Block diagram of a proposed parallel-pipelined deblocking filter with adaptive power

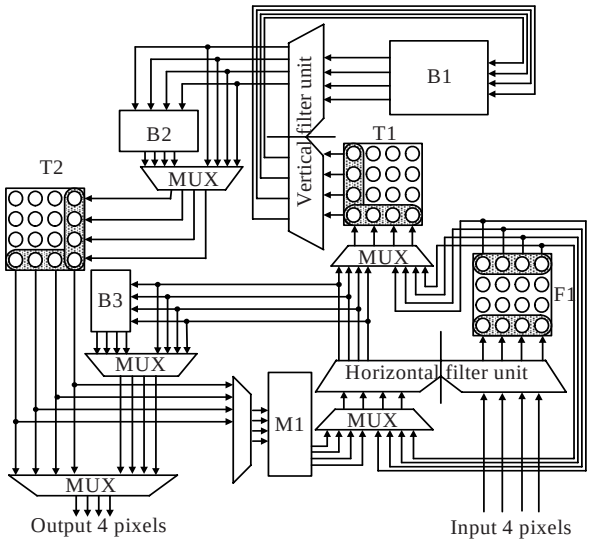


Fig.8 Architecture of the parallel-pipelined filter unit

As shown in Fig.8, the horizontal LOP filter (HLF) and vertical LOP filter (VLF) can both filter 8 pixels every line in a cycle. F1 is a 4×32-bit FIFO to buffer the samples in a 4×4 block row by row. B1 is a 16×32-bit two-port SRAM, used to buffer the samples input to the VLF. T1 and T2 are two 4×4 transpose register arrays, transposing a block in 4 cycles (Huang *et al*., 2005). B2 is a 12×32-bit two-port SRAM and B3 is a 16×32-bit two-port SRAM. B2 and B3 are used to buffer the blocks to match the timing of output. M1 is a 32×32-bit two-port SRAM used to buffer the leftmost 4 luma and 4 chroma blocks for deblocking. Table 2 shows the data flow of deblocking the MB in Fig.6 when the gated clock is enabled.

When the gated clock is enabled, the non-skipped blocks are sent to the filter unit in a raster scan order. Each block needs 7 block cycles (28 cycles in total) to process, where the blocks are then output in the same order as input. During the process, some blocks—if finishing the deblocking ahead of time—will be buffered until the right timing of output. With the block-based pipeline, an MB can be processed in 4*n* cycles (*n* is the number of non-skipped blocks). When being disabled, the gated clock takes a block cycle to move each skip-block from the reconstructed MB buffer to the off-chip memory bus or to M1 (the skip-blocks on the right boundary of the MB are loaded from the reconstructed MB buffer to M1). The clock to M1 is not gated because M1 can be accessed when the gated clock is disabled.

We can find from Table 2 that, the free space of B1 can just be used to buffer the blocks in B2, and the free space of M1 can coincidentally be used to buffer the blocks in B3. Two quad-port memories (two-port read and two-port write) Q1 and Q2 are therefore designed to substitute B1, B2, M1, and B3. The quad-port memory is made up of 4 two-port SRAMs (4×32 bits for Q1 and 8×32 bits for Q2) and some multiplexers (Fig.9).

SIMULATION RESULTS AND COMPARISON

In order to test the performance of the proposed architecture, we implemented the architecture in Verilog and synthesized it using the Synopsys Design

Table 2 Data flow of the pipelined deblocking process for the MB in Fig.6*

Block cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Input block	Y3	Y4	Y7	Y8	Y11	Y12	Y13	Y14	Y15	Y16	Y18	Y19	Y20	U22	U24	U25
F1		Y3	Y4	Y7	Y8	Y11	Y12	Y13	Y14	Y15	Y16	Y18	Y19	Y20	U22	U24
T1			*Y3*	*Y4*	*Y7*	*Y8*	*Y11*	*Y12*	*Y13*	*Y14*	*Y15*	*Y16*	*Y18*	*Y19*	*Y20*	U22
B1							***Y8***	***Y8***			***Y14***	***Y14***	***Y14***		***Y19***	***Y19***
						Y7	***Y7***			***Y13***	***Y13***	***Y13***	***Y13***	***Y18***	***Y18***	***Y18***
					Y4	***Y4***			***Y12***	***Y12***	***Y12***	***Y12***	***Y16***	***Y16***	***Y16***	
				Y3	***Y3***			***Y11***	***Y11***	***Y11***	***Y11***	***Y15***	***Y15***	***Y15***		***Y20***
M1	**B1**	**B1**	**B1**									**Y8**	**Y8**	**Y8**	**Y8**	**Y8**
	B2	**B2**	**B2**	**B2**	**B2**									**Y12**	**Y12**	**Y12**
	B3	B3	B3	B3	B3	B3	B3									
	…	**B4**	**B4**	**B4**	**B4**	**B4**	**B4**	**B4**	**B4**	**B4**	**B4**					
	B8	**…**	**…**	**…**	**…**	**…**	**…**	**…**	**…**	**…**	**…**	**…**	**…**	**…**	**…**	**…**
B2								***Y7***	***Y7***						***Y15***	
							Y4	***Y4***						***Y14***		
						Y3	***Y3***		***Y8***	***Y8***						***Y16***
B3												**B4**	**B4**	**B4**	**B4**	**B4**
								B3	**B3**	**B3**	**B3**	**B3**	**B3**	**B3**	**B3**	**B3**
						B2	**B2**	**B2**	**B2**	**B2**	**B2**	**B2**	**B2**			
				B1	**B1**	**B1**	**B1**	**B1**	**B1**	**B1**	**B1**					**B5**
T2								**Y3**	**Y4**	**Y7**	**Y8**	**Y11**	**Y12**	**Y13**	**Y14**	**Y15**
VFU input	Y3	Y4	Y7	Y8	Y11	Y12	Y13	Y14	Y15	Y16	Y18	Y19	Y20	U22	U24	U25
			B1	Y7	B2	Y11	B3	Y13	Y14	Y15	B4	Y18	Y19		B5	B6
HFU input			*Y3*	*Y4*	*Y7*	*Y8*	*Y11*	*Y12*	*Y13*	*Y14*	*Y15*	*Y16*	*Y18*	*Y19*	*Y20*	*U22*
					Y3	*Y4*	*Y7*	*Y8*			*Y11*	*Y12*	*Y14*	*Y15*	*Y16*	
Output block								**Y3**	**Y4**	**Y7**	**B1**	**Y11**	**B2**	**Y13**	**Y14**	**Y15**

* Gated clock enabled. Bold fonts denote that the block have finished deblocking; italic fonts indicate that the block samples are stored in columns. VFU: vertical filter unit; HFU: horizontal filter unit

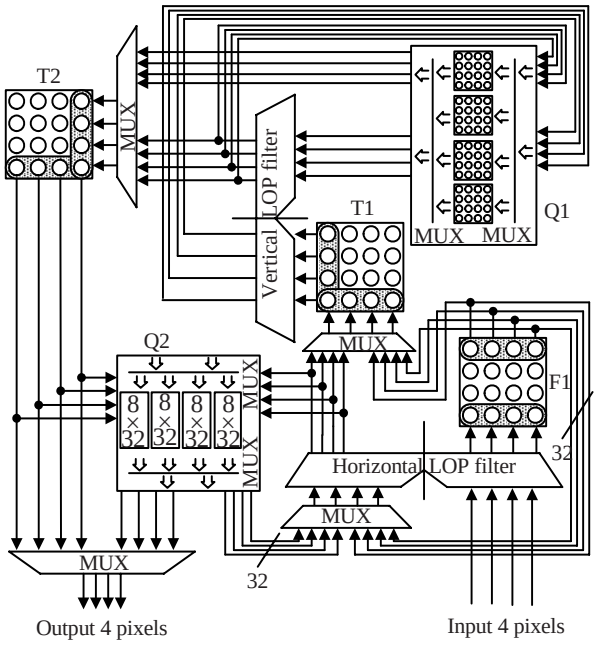


Fig.9 Architecture of a proposed parallel-pipelined filter unit with quad-port memories

Compiler. The hardware implementation took 19.8k gates under the TSMC 0.13 μm standard cell library (provided by ArtisanR®) at 100 MHz.

Table 3 compares our architecture with the previous designs. With the block based pipeline, our architecture took (20+12)×4=128 cycles at most to perform the deblocking for an MB, significantly outperforming the previous designs by 1.25~4.8 times (Vladimir's design excluded). The maximum number of off-chip memory access times was 144 (read 0.25 MB and write 1.25 MB) in our design, including the necessary memory accesses (96 times per MB) to store the reconstructed MB to the off-chip frame buffer. Therefore, the increased number of off-chip memory access times for the deblocking process was only 144−96=48.

In the worst cases, the design of (Chen and Chen, 2006) can work faster than the proposed architecture at the cost of using a FIFO with a size of 44 blocks. And the MBs in Chen's design are filtered column by column, which breaks the scan order of MB decoding. It means that the deblocking process cannot start until

Table 3 Performance comparison between the proposed design and the previous designs

Design	Architecture	Cycles per MB	Memory accesses per MB		Gate count
			On-chip	Off-chip*	
Huang *et al*., 2003	Dual arrays+Dual-port SRAM	614	512	320	20.66k
Li *et al*., 2005	Dual-port SRAM (8×80×8 bits)	566	384	320	9.57k
Zhao and Jiang, 2007	Single-port SRAM (16×32 bits)	352	320	320	–
Chen and Chen, 2005	Dual-port SRAM (32-bit) or two single-port SRAMs	240	320	320	22k
Xu and Choy, 2008	5-stage pipeline; single-port SRAM (2×96×32+2*N*×32 bits)	204	192	192	21.5k
Vladimir and Ivan, 2007	Dual-port SRAM (8×32 bits)	7552 (4480)#	512	512	1.78k
Min and Chong, 2007	Two-port SRAM (16×32 bits)	232	320	320	
Hao and Martin, 2008	Dual-port SRAM (16×32+*N*×32 bits); shift register array (2×64×32 bits)	192	192	192	10.46k
Liu *et al*., 2007	Dual-port RAM (2×96×32+2(*N*+12)×32 bits)	243	192	192	21.1k
Zhao and Lu, 2007	Register arrays (6×16×8 bits); two edge filters	172	320	128	–
Chen and Chen, 2007	Dual-port SRAM (24×32 bits); two edge filters	160	256	256	20.84k
Chen and Chen, 2006	Dual-port SRAM (180×32 bits, QCIF); two edge filters	100	192	192	14.75k
Proposed	Two-port SRAM (52×32 bits); two edge filters	128 (0)#	240 (0)#	144 (96)#	19.8k

* Assume loading MB from off-chip buffer as in (Huang *et al*., 2003). # maximum (minimum). *N* is frame width in pixels

all the samples in the frame have been reconstructed, and hence the decoding delay and the off-chip memory accesses will be increased (the size of the reconstructed frame is too large for on-chip memory).

In the best situation, all blocks in the MB can be skipped. The gated clock can be disabled all the time. The skip-blocks are moved from the reconstructed MB buffer to the off-chip memory bus directly, and the leftmost 4 luma and 4 chroma blocks are transferred from buffer M1 to the off-chip memory bus. No additional off-chip memory access is needed at this time.

In the experiments, we simulated the average power of deblocking filters using Synopsys Primepower. The simulation results of the proposed architecture are shown in Fig.10 with comparison to a previous design (Chen and Chen, 2006), which also uses two edge filters. Three test videos (Foreman, Mobile, News on CIF format, 30 frames/s, P frame in a row except for the first frame) were adopted for power simulation. The proposed architecture and Chen's architecture worked at the same voltage (1.2 V) and the lowest frequency (1.91 MHz for Chen's design, 1.52 MHz for the proposed architecture).

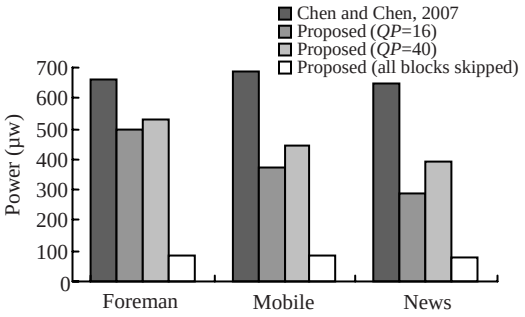


Fig.10 Power simulation for the proposed architecture

Also, we simulated the minimal power of the deblocking filter by forcing all blocks to be skipped, when the average power was reduced by up to about 89%. It can be seen that the power was adaptively reduced for different videos with the proposed architecture.

CONCLUSION

We study the adaptivity of the H.264 deblocking filter and propose a parallel-pipelined architecture of

the H.264 deblocking filter with adaptive power. The vertical and horizontal edges are simultaneously processed in an advanced scan order to reduce deblocking cycles and memory accesses. The proposed architecture is very appropriate for real-time deblocking of HDV (1920×1080 pixels/frame, 60 frames/s video signals) video operation at 62 MHz. Moreover, the proposed architecture can reduce the dynamic power adaptively for different videos. Therefore, with the fast deblocking speed, low off-chip memory access, and low dynamic power, the proposed architecture can work better on a mobile or PDA platform.

References

Arbelo, C., Kanstein, A., Lopez, S., 2007. Mapping Control-intensive Video Kernels onto a Coarse-grain Reconfigurable Architecture: The H.264/AVC Deblocking Filter. IEEE Conf. on Design, Automation and Test, p.1-6. [doi:10.1109/ DATE.2007.364587]

Chen, C.M., Chen, C.H., 2005. An Efficient Architecture for Deblocking Filter in H.264/AVC Video Coding. IASTED Int. Conf. on Computer Graphics and Imaging, p.177-181.

Chen, C.M., Chen, C.H., 2006. Window Architecture for Deblocking Filter in H.264/AVC. IEEE Int. Symp. on Signal Processing and Information Technology, p.338-342. [doi:10.1109/ISSPIT.2006.270822]

Chen, C.M., Chen, C.H., 2007. An efficient pipeline architecture for deblocking filter in H.264/AVC. *IEICE Trans. Inf. Syst.*, **E90-D**(1):99-107. [doi:10.1093/ietisy/e90-1.1.99]

Cheng, C.C., Chang, T.S., 2005. A Hardware Efficient Deblocking Filter for H.264/AVC. IEEE Conf. on Consumer Electronics, p.235-236. [doi:10.1109/ICCE.2005.1429804]

Hao, W.N., Radetzki, M., 2008. A Data Traffic Efficient H.264 Deblocking IP. IEEE Int. Symp. on Circuits and Systems, p.3430-3433. [doi:10.1109/ISCAS.2008.4542196]

Huang, Y.W., Chen, T.W., Hsieh, B.Y., Wang, T.C., Chang, T.H., Chen, L.G., 2003. Architecture Design for Deblocking Filter in H.264/JVT/AVC. IEEE Conf. on Multimedia and Expo, p.693-696. [doi:10.1109/ICME.2003.1221012]

Huang, Y.W., Hsieh, B.Y., Chen, T.C., Chen, L.G., 2005. Analysis, fast algorithm, and VLSI architecture design for H.264/AVC intra frame coder. *IEEE Trans. Circuits Syst. Video Technol.*, **15**(3):378-401. [doi:10.1109/TCSVT.2004.842620]

Li, L.F., Goto, S., Ikenaga, T., 2005. A highly parallel architecture for deblocking filter in H.264/AVC. *IEICE Trans. Inf. Syst.*, **E88-D**(7):1623-1629. [doi:10.1093/ietisy/e88-d.7.1623]

Liu, T.M., Lee, W.P., Lee, C.Y., 2007. An in/post-loop deblocking filter with hybrid filtering schedule. *IEEE Trans. Circuits Syst. Video Technol.*, **17**(7):937-943. [doi:10.1109/TCSVT.2007.897467]

Lou, J., Jagmohan, A., He, D.K., Lu, L.G., Sun, M.T., 2007. High Speed H.264 High Profile Deblocking Using Statistical Analysis and Logic Optimization. Int. Conf. on Multimedia and Expo, p.1918-1921. [doi:10.1109/ICME.2007.4285051]

Min, K.Y., Chong, J.W., 2007. A Memory and Performance Optimized Architecture of Deblocking Filter in H.264/AVC. IEEE Conf. on Multimedia and Ubiquitous Engineering, p.220-225. [doi:10.1109/MUE.2007.21]

Sullivan, G., Topiwala, P., Luthra, A., 2004. The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions. SPIE, **5558**:454. [doi:10.1117/12.564457]

Vladimir, C., Ivan, M., 2007. Area-time Tradeoffs in H.264/AVC Deblocking Filter Design for Mobile Devices. ISSPA, p.1-4. [doi:10.1109/ISSPA.2007.4555338]

Warrington, S., Shojania, H., Sudharsanan, S., 2006. Performance Improvement of the H.264/AVC Deblocking Filter Using SIMD Instructions. IEEE Proc. ISCAS, p.2697-2700. [doi:10.1109/ISCAS.2006.1693180]

Xu, K., Choy, C.S., 2008. A five-stage pipeline, 204 cycles/MB, single-port SRAM-based deblocking filter for H.264/AVC. *IEEE Trans. Circuits Syst. Video Technol.*, **18**(3):363-377. [doi:10.1109/TCSVT.2008.918437]

Zhao, S., Lu, C., 2007. VLSI Design for De-blocking Filter of H.264 Decoder. IEEE Conf. on ASIC, p.786-789. [doi:10.1109/ICASIC.2007.4415748]

Zhao, Y.X., Jiang, A.P., 2007. An effective parallel processing architecture for deblocking filter in H.264. *Acta Sci. Nat. Univ. Pekin.*, **43**(5):649-653 (in Chinese).