An In/Post-Loop Deblocking Filter With Hybrid Filtering Schedule

Tsu-Ming Liu, *Student Member, IEEE*, Wen-Ping Lee, and Chen-Yi Lee, *Member, IEEE*

***Abstract*—In this paper, we propose a high-throughput deblocking filter to perform the in-loop or post-loop filtering process for different standard requirements. The performance improvement is very mild if we replace a post-loop filter with an in-loop filter. To alleviate this problem, we derive an integration-oriented algorithm that can be reconfigured as the in-loop or post-loop filter. Moreover, we develop a hybrid filtering schedule to reach a lower bound of processing cycles. In particular, we reschedule the filtering order and reuse the intermediate pixels when the deblocking filter switches the filtered edges from vertical to horizontal direction. Finally, a 0.18-μm CMOS design that performs the in/post-loop filter with the hybrid filtering schedule is implemented. The synthesized gate counts are 21.1 K which is reduced to 70% of preliminary design that performs the in-loop or post-loop filter separately. Moreover, it achieves 4×10^5 macroblock/s of throughput rate at a 100-MHz clock rate.**

***Index Terms*—Deblocking filter, H.264/AVC, high-throughput, MPEG-4.**

I. Introduction

ALL current video compression standards including MPEG-1/2/4, H.261/2/3/4, AVS and VC-1 [1] perform a block-based discrete cosine transform, quantization, and motion compensated prediction to improve the coding efficiency. Nevertheless, the quantization errors bring the annoying discontinuity on each block boundary. Hence, a deblocking filter is required to remove this discontinuity and improve the visual quality. Among various video standards, the deblocking filter modules can be divided into two classes: in-loop and post-loop filters. For instance, an in-loop filter [3] is standardized by newly-announced H.264/AVC while a post-loop filter [2] can be applied to prevalent MPEG-x family for improving visual quality. However, considering the multi-standard integration, replacing the post-loop filter with the in-loop filter will degrade the visual quality. Instead, we combine the in-loop and post-loop filters in algorithmic and architectural levels to save cost and improve subjective and objective visual quality.

Several deblocking filters [4], [5] have already appeared since it becomes one of performance bottlenecks at the decoding side. Though these techniques carry out efficient architectures, they follow the standard-defined filtering order, leading to additional cycles required when the deblocking filter switches filtered edges from vertical to horizontal direction. Sheng *et al.* [6] proposed a 2-D processing order to reschedule the filtering order and reduce the processing cycles, but this order introduces large storages (eight 4×4 buffers). In our design, we develop a hybrid filtering order to reuse the pixel without affecting the standard-defined data dependency. It not only eliminates the processing cycles when switching the filtered edges but reduces the intermediate buffer cost. Finally, a 0.18-μm CMOS design of the deblocking filter is implemented. The processing throughput achieves 4×10^5 macroblock (MB)/s at 100 MHz and is 1.5–2.5 times larger than that of existing designs [4]–[6]. The rest of this paper is organized as follows. Section II develops an in/post-loop filtering algorithm. Section III presents the associated architecture with the hybrid filtering schedule. Section IV describes the simulation and implementation results. Finally, concluding remarks are made in Section V.

Manuscript received April 25, 2005; revised November 19, 2005. This paper was recommended by L.-G. Chen.

The authors are with the Department of Electrical Engineering, National Chiao-Tung University, Hsinchu 30050, Taiwan, R.O.C. (e-mail: mingle@si2lab.org; mb@si2lab.org; cylee@si2lab.org).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2007.897467

II. In/Post-Loop Deblocking Algorithm

A. Algorithmic Preview

Due to a great diversity of deblocking filters in different standards, we tabulate each feature in Table I. The filtering control decides the filtering order and the size of filtered boundaries. In general, most of deblocking filters obey an order that performs on the horizontal edges and follows by the vertical edges. But, this order is different from that in H.264/AVC. As for the filtered boundary, the in-loop filter of H.264/AVC is applied to each edge of 4×4 subblock while the post-loop filter is performed on the boundaries of 8×8 block.

Filtering processes can be divided into three main parts. The first part of processes is the strength decision. It governs the filtering intensity in that edge. H.264/AVC employs a boundary strength (bS) (i.e., bS spreads from 0 to 4, 5-strength) to calculate the strength in each filtering mode. VC-1 adopts the edge_strength (i.e., only true or false, 2-strength) to realize the strength decision [7]. Moreover, MPEG-4 and H.263 are 2-strength (i.e., $\mathrm{eq}_{\mathrm{cnt}} >= 6$ or $\mathrm{eq}_{\mathrm{cnt}} < 6$) and 12-strength (i.e., strength= 1–12) respectively. The second part of processes is the mode decision which is comprised of strong and weak modes. For instance, in MPEG-4, Kim *et al.* [2] exploited smooth regions and default modes as strong and weak modes, respectively. List *et al.* [3] applied strong and weak modes when the bS is equal to or less than 4 respectively. A third part of filtering processes is the edge filter. The numbers of input pixels are related to the filtering performance as well as computational complexity. After previewing aforementioned features, we conclude that there are great diversities in those filters. Hence, a combined in/post-loop filter algorithm is of great challenge for saving cost.

B. In/Post-Loop Algorithm

Using a single algorithm to realize in-loop or post-loop filter is inferior since the source of blocking artifacts comes from a distinct quantization process, IDCT kernel and the motion compensated algorithm. From the experimental results, the quality improvement is very mild (only 0.04 dB) when we

TABLE I
FEATURES OF DEBLOCKING FILTER IN DIFFERENT STANDARDS

Deblocking Filter		In-loop Filter		Post-Loop Filter	
Standardization		Normative		Informative	
STANDARD		H.264/AVC	VC-1	MPEG-4(Annex F.3)	H.263(Annex J)
Filtering Control	Filtering Order	Vertical First	Horizontal First	Horizontal First	Horizontal First
	Filtered Boundary	4×4	4×4/4×8/8×4/8×8	8×8	8×8
Filtering Process	Strength/Mode	5/2	2/1	2/2	12/1
	No. of input pixels	8	8	10	4

replace a post-loop filter with an in-loop filter. To alleviate this problem, we propose an integration-oriented algorithm which tightly combines H.264/AVC in-loop filter with MPEG-4 post-loop filter. Specifically, we keep the filtered boundaries of 4×4 and 8×8 in the in-loop and post-loop filters respectively. Additionally, to unify into a single architecture, the filtering order in post-loop filters has been changed from horizontal to vertical edges first. With regard to filtering processes, a triple-mode decision and triple pixel-in-pixel-out edge filter are proposed to improve the integration efficiency. Moreover, they provide an easy exchange of different filter types without changing a hardware prototype.

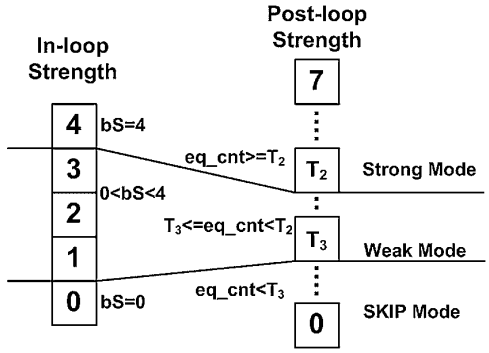


Fig. 1. Triple-mode decision of the in/post-loop filter.

C. Triple-Mode Decision

A triple-mode decision adopts a SKIP mode and resource sharing technique to reduce filtering complexity and integration cost respectively. Firstly, this decision has been applied to H.264/AVC and employs strong, weak and SKIP modes according to the bS. As to the post-loop filter in MPEG-4, Kim *et al.* [2] exploited the threshold T_2 as 6 to distinguish between default (i.e., weak) and dc offset (i.e., strong) modes. However, it is very time-consuming because there is no skip conditions applied and all 8×8 edge boundaries perform filtering processes. To alleviate this problem, we introduce another threshold T_3 to reduce the computation in Fig. 1. Moreover, since fixed thresholds $\{T_2, T_3\}$ cannot achieve better performance, we use the side-information (e.g., MVD, CBP, MB_Type) to adjust the thresholds dynamically. In Table II, we propose a compound decision method to share the hardware resource since MPEG-4's $\{T_2, T_3\}$ are similar to H.264's $\{\mathrm{bS}, \alpha, \beta, t_{C0}\}$. Moreover, we found that different bit rates contribute to the difference of the threshold T_2. Introducing a term of t_{T2} as a function of quantization parameter (QP) makes it more robust in terms of the bit rate variations. In conclusion, the proposal reduces not only the computation through the SKIP mode but also the integration cost by the compound method.

D. Triple Pixel-in-Pixel-Out (P-i-P-o) Edge Filter

We develop a triple P-i-P-o edge filter to reduce the integration cost. In the post-loop mode, the edge filter retains the default mode and discards the dc offset mode because the default mode is of the prime concern while the dc offset mode is broadly similar to the strong mode of the in-loop filter. That is, we can replace the edge filter of dc offset mode with that of "bS = 4" (strong mode) for an integration-oriented design approach. We change the approximated discrete cosine transform (DCT) kernel (i.e., [2 -5 5 -2]) to [2 -4 4 -2]. As a result, we make use of shifters instead of constant multipliers. Moreover, to merge the edge filter in the weak mode, we modify the numbers of input pixels to 8 pixels in the post-loop filter. Thus, the numbers of input pixels in the in-loop and post-loop filters are equivalent. In conclusion, three data flows (i.e., strong, weak and SKIP) and related pseudo codes are highlighted in Fig. 2, and some modifications are made on the post-loop filter to improve the integration efficiency. These modifications definitely reduce the integration overhead with a penalty of slight performance loss. This loss will be further addressed in Section IV-A.

III. HIGH-THROUGHPUT IN/POST-LOOP ARCHITECTURE

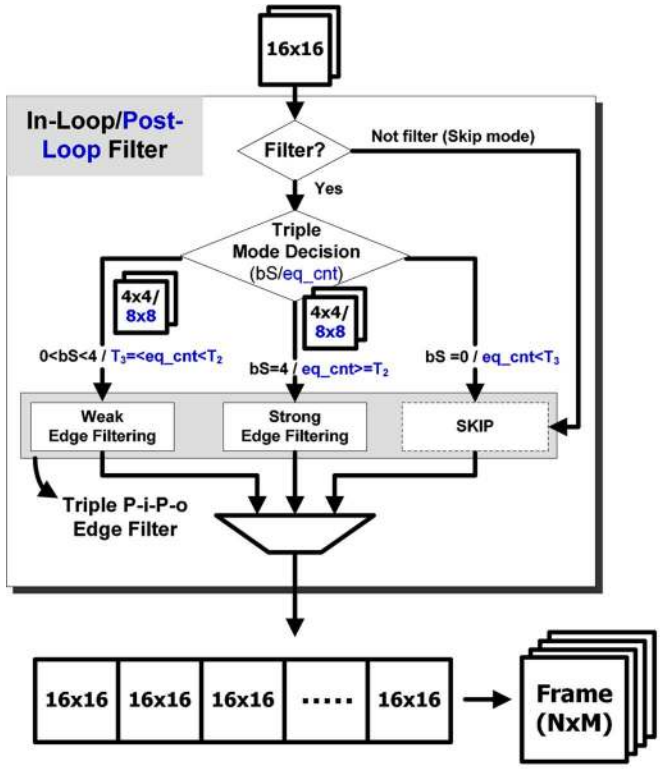
This section presents a high-throughput architecture with the hybrid filtering schedule. The associated block diagram is depicted in Fig. 3. Two dedicated single-port SRAMs (content and slice) are designed to not only store the current and neighboring pixels but also achieve an efficient data access. Furthermore, we propose the hybrid filtering schedule and introduce four 4×4 pixel buffers to reduce the numbers of processing cycles.

A. Memory Organization

The proposed memory organizations are twofold; content and slice memory. The address depth is decided by the YUV formats (4:4:4, 4:2:2, or 4:2:0), and the data word size is based on the 32-bit of the column-of-pixel (CoP) [see Fig. 4(a)] for the reduced memory accesses in the intra/inter prediction unit [8]. The content memory stores the unfiltered pixels prior to the deblocking filter. Moreover, it adopts the ping-pong structure and stores two MBs to resolve the structural hazard when reading and writing processes occur simultaneously. Hence, the content memory is of size $(16 + 8) \times 4 \times 32 \times 2$ (in 4:2:0). On the other hand, the slice memory stores the upper and left neighboring pixels for follow-up filtering processes. Considering a

TABLE II
COMPOUND METHOD FOR THE STRENGTH DECISION

In/Post-loop Decision Method		In-loop	Post-loop	
Block modes and conditions	**Semantics**	**bS**	**T_3**	**T_2**
One of the blocks is intra and the edge is a macroblock edge	MB_TYPE==INTRA && MB Edge	4	T_{3i}*	T_{2i}*
One of the blocks is intra	MB_TYPE==INTRA	3	T_{3i}	T_{2i}
One of the blocks has coded residuals	CBP != 0	2	T_{3i}+1	T_{2i}+1
Difference of block motion $\geq$ 4 at luma sample difference	$MVD_x \geq 4$ || $MVD_y \geq 4$	1	T_{3i}+2	T_{2i}+1
Else	Else	0	T_{3i}+2	T_{2i}+1+t_{T2}(QP)



(a)

```
// Step1: In/Psot-Loop recognition from bit-stream syntax
in/post = (flag) ? In_loop : post_loop;  // in or post-loop configuration
In/Post-Loop De-blocking Filter(16x16 MB, in/post)
{
 if(de-blocking_assert)
 {
    filter_end = 0;
    while (filter_end==0)
    {
       // Step2: Filtering Control
       horizontal first, followed by vertical···
       edge_detection = 4x4 or 8x8

       // Step3: Strength Calculation
       strength = (in/post) ? Calculate_bS() : Calculate_eq_cnt();

       // Step4: Mode Decision
       mode = Triple_Mode_Decision(strength);

       // Step5: Edge Filtering
       if (mode==strong)       result = Strong_Filtering();
       else if (mode==weak)  result = Weak_Filtering();
       else if (mode==SKIP)  SKIP applied;

       // Step6: Return the filtered pixel and terminate
       write the filtered result into 16x16 MB;
       if (end_of_de-blocking)  filter_end = 1;
    }
 }
 else
    SKIP applied;
}
```

(b)

Fig. 2. (a) Data flow and (b) pseudo-code of the in/post-loop algorithm.

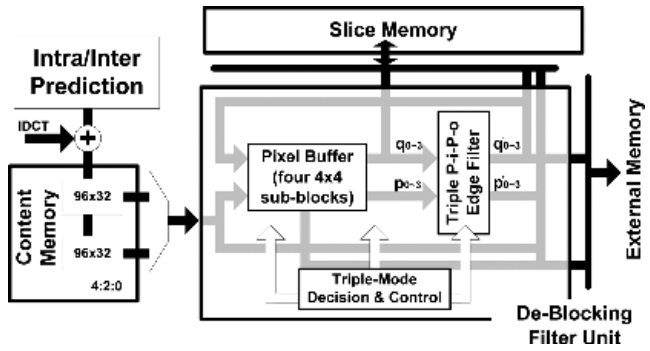


Fig. 3. Block diagram of in/post-loop dedblocking filter.

frame size of $N \times M$ in Fig. 4(b), each square represents the 16×16 MB. Each MB contains 16 points and 4×4 pixels within each point. When filtering processes are performed from the MB index B to $B+1$, upper and left neighbors will update the pixel values as the arrows indicate. The shaded region should be kept when the filtering index is $B+1$. Therefore, the size of slice memory is $2 \times (\mathrm{N}+12) \times 32$ for the 4:2:0 format.

B. Hybrid Filtering Schedule

We propose a hybrid filtering schedule to reuse the intermediate data and thereby eliminate the additional memory accesses when deblocking filter changes the filtered edges from vertical to horizontal direction. Fig. 5(a) describes the standard-defined filtering orders where vertical edges are filtered first, followed by horizontal edges. However, a main drawback of this direct approach is that the intermediate data have to be written into the internal memory in one direction and then read again in another direction. For instance, considering the black region in Fig. 5(a),

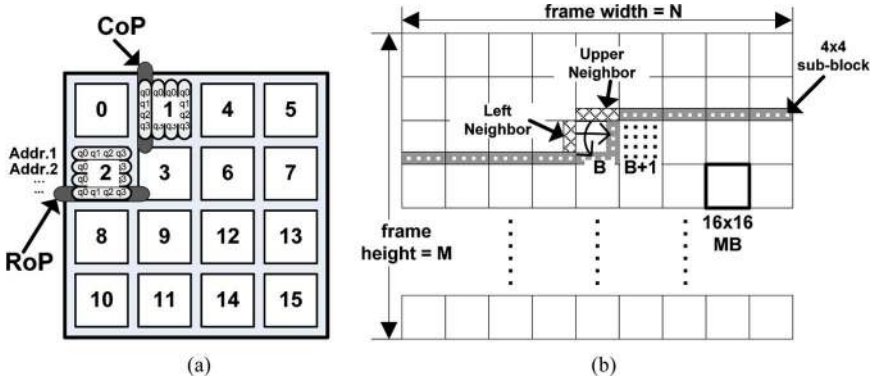


Fig. 4. Data organizations of (a) content and (b) slice memory.

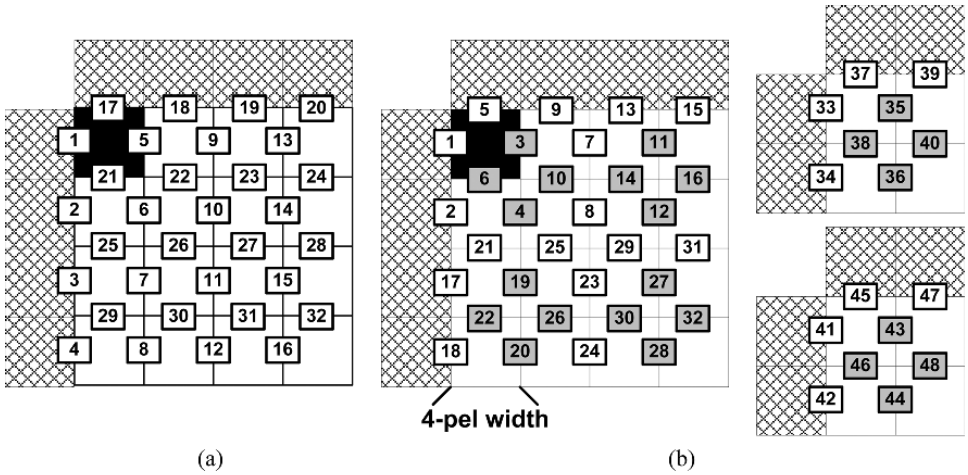


Fig. 5. (a) Standard-defined and (b) proposed schedule.

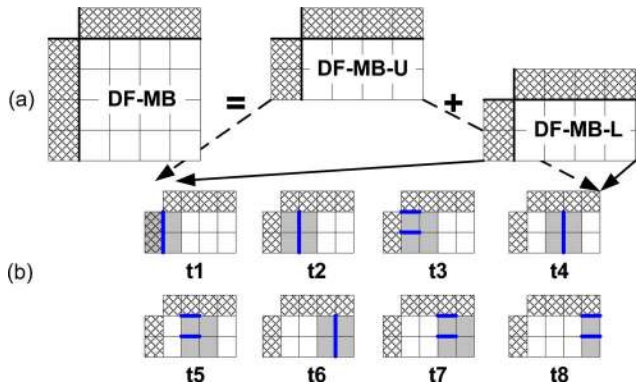


Fig. 6. (a) Partitioned MB and (b) each time index for the hybrid filtering schedule.

the edge #1 will be filtered first, followed by the edge #5. After that, the processing data in the black region cannot be reused since the filtering orders between vertical and horizontal edges become longer (i.e., #5 versus #17). Therefore, the memory accesses are required in both vertical and horizontal directions. To alleviate this problem, we develop a hybrid filtering schedule without affecting the standard-defined data dependency in Fig. 5(b) and all unshaded numbers are performed in 8×8 post-loop filters. Considering the orders in the black region to perceive a contrast, the black region can be reused because the orders between different directions become close. Therefore, the proposal prevents the data re-access for different directions and reuses the intermediate pixels to reduce the processing cycles.

Though Sheng *et al.* [6] proposed a novel schedule to reduce the processing cycles, this schedule requires eight 4×4 sub-blocks as the kept buffers. To reduce this buffer size, each MB can be partitioned into two main parts (i.e., *Deblocking Filter-MB-Upper* or *Lower*) in Fig. 6(a), and each part is composed of eight time indexes to perform the filtering procedure in Fig. 6(b). Each bold line represents the edge to be filtered in the corresponding time index. As a result, our kept buffer size is four 4×4 sub-blocks where is located on shaded regions. By the same way, the proposed schedule is performed on the chroma MB as well.

C. High-Throughput Architecture

The high-throughput architecture with hybrid filtering schedule is presented in this subsection. The detailed signal flow of Fig. 3 has been redrawn in Fig. 7. Specifically, all signals are 32-bit wide and possess the CoP organization. The signal names represent the writing/reading to/from the storage modules including the slice, content, and external memory. Accordingly, the behavior of Fig. 7 can be divided into two main parts: writing processes $\{wt\_\text{Slice}\_0 \sim 2, wt\_\text{EXT}\_0 \sim 1, wt\_B\_0 \sim 3\}$ and reading processes $\{rd\_\text{Slice}\_0 \sim 1, rd\_C\_0, rd\_B\_0 \sim 2\}$.

The key idea of the high-throughput architecture is that the content memory is exploited only for the reading processes. In Fig. 7, the writing signal, wt_Slice_0, is activated on the edges

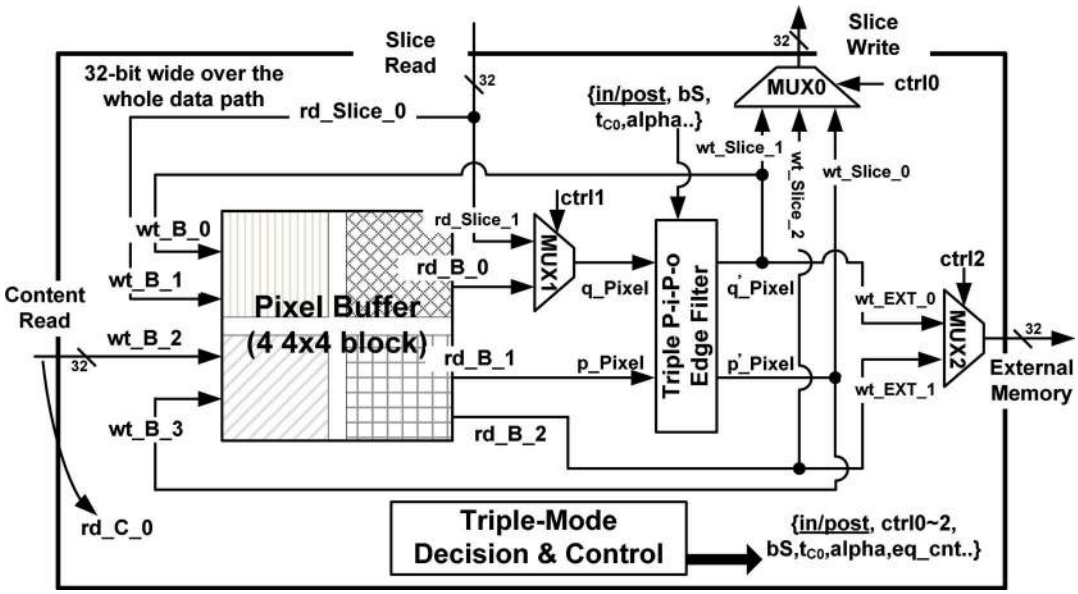


Fig. 7. Detailed architecture for the in/post-loop deblocking filter.

TABLE III
VALUES OF t_{T2} WITH A FUNCTION OF QP

QP	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
t_{T2}	2	2	1	1	1	1	1	1	0	0	0	0	0	0	0	0
QP	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
t_{T2}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TABLE IV
POST-LOOP FILTERING PERFORMANCE IN TERMS OF LUMA PSNR

MPEG-4 Decoder		PSNR-Y [dB]		
Bit Rate	CIF Sequence	w/o filter	MPEG-4 Annex F.3 filter	Proposed
150kbps	Table	32.01519	32.13722	32.13138
	Mobile calendar	24.97704	25.00832	25.04150
	Mother & daughter	38.73812	39.00201	39.02740
	Stefan	27.02727	27.14439	27.10813
450kbps	Table	37.11400	37.18894	37.19088
	Mobile calendar	27.43971	27.49859	27.48667
	Mother & daughter	42.85162	42.99503	42.99579
	Stefan	30.63070	30.77906	30.72632
1500kbps	Table	42.83698	42.84450	42.87695
	Mobile calendar	34.50489	34.57378	34.56172
	Mother & daughter	46.33545	46.48835	46.44548
	Stefan	38.36032	38.48165	38.44038

6, 10, 14, and 16 because the lower subblocks become the upper neighboring subblocks of $DF - MB\_L$ [see Fig. 6(a)]. Therefore, the wt_Slice writes the filtering results into slice memory for follow-up filtering processes. For the wt_EXT, it writes filtered data into the external memory. On the other hand, the reading signal, $rd\_\mathrm{Slice\_0}$, is activated on vertical edges while the $rd\_\mathrm{Slice\_1}$ is valid on horizontal edges. In addition, the rd_C_0 directly feeds through the pixel buffers. In other words, content memory is employed for the reading processes, and there is no need to write the filtered results into the content memory in one direction and thereby read them in another direction. Therefore, the proposal exploits four 4×4 buffers to reuse the intermediate pixel and eliminate the writing accesses of content memory.

IV. SIMULATION AND IMPLEMENTATION RESULTS

A. Performance Evaluation

The modifications of the post-loop filter improve the integration efficiency at a cost of slight performance degradation. For the experiments of MPEG-4's post-loop filter, the thresholds of $T_{2i} = 5$ and $T_{3i} = 0$ (see Table II) are employed without loss of generality. Further, we adopt Table III as the induced term of t_{T2}. QP stands for "quantizer precision," and we use 5-bit as a default value that ranges from 0 to 31. All alterations of the MPEG-4's post-loop algorithm have been addressed in Section II, and specific results are given in Table IV. All sequences are defined in CIF (352×288) and intra-period 15 with 30 fps

(a) (b) (c)

Fig. 8. Subjective quality comparison between the post-loop filter and proposed design. (a) Without filter. (b) MPEG-4's filter. (c) Proposed.

TABLE V
CYCLE ANALYSIS OF THE DEBLOCKING FILTERS

Cycle Counts			[4][1]	[5][2]	[6]	Proposed
Vertical / Horizontal			Separate	Separate	Hybrid	Hybrid
Pre-process (Initial) stage			160	N/A	64	0
Filter-process Stage	Luma	Horizontal	128	106	128[3]	148
		Vertical	200	106		
	Chroma	Horizontal	64	74	64[3]	88
		Vertical	112	74		
Post-process (write-back) stage			160	N/A	160	0
Misc.			54	N/A	30	7
Total			878	360+N/A	446	243

[1]: We list the single-port memory configuration for a fair comparison.

[2]: We only consider the worst case and exclude the effect of mode decision for a fair comparison.

[3]: Authors didn't report the processing overheads between the internal memory and kept buffers.

throughout 300 frames. We show that the performance degradation is less than 0.05 dB as compared to the MPEG-4's post-loop filter [2]. From the subjective point of view, we capture the 20th frame to give a comparison in Fig. 8.

B. Processing Cycle Analysis

To clarify the cycle reduction, we formulate processing cycles in (1) and (2) where C.C. means cycle counts. The overall cycles of deblocking filters can be considered as a combination of the pre-process, filter-process and post-process. The pre-process is an initial stage which loads external data (neighboring pixel) into slice memory while the post-process is a write-back stage which writes filtered results from slice memory to external memory. In the filter-process, the processing cycles include slice or content Memory to pixel Buffers (i.e., $M_{\text{slice/content}}$-to-B_{pixel}), generic filtering, and pixel Buffers to slice Memory (i.e., B_{pixel}-to-M_{slice}). The processing cycles of the generic filtering are $4\times(32+16)$ which become a lower bound to fulfill filtering processes if the rest of processing cycles are zero in an ideal case.

$$\begin{aligned}
&\text{Total Cycle Counts}\\
&\quad=\text{C.C.}_{\text{Pre-process}}+\text{C.C.}_{\text{FilterProcess}}\\
&\qquad+\text{C.C.}_{\text{Post-Process}}+\text{C.C.}_{\text{misc.}}\\
&\text{C.C.}_{\text{Pre-process}}\\
&\quad=\text{C.C.}_{\text{initial}}=\text{C.C.}_{\text{Mexternal-to-Mslice}}\\
&C.C_{\text{Post-Process}}\\
&\quad=\text{C.C.}_{\text{write back}}\\
&\quad=\text{C.C.}_{\text{Mslice-to-Mexternal}} \qquad (1)
\end{aligned}$$

$$\begin{aligned}
&\text{C.C.}_{\text{Filter-Process}}\\
&\quad=\text{C.C.}_{\text{Luma Filter}}+\text{C.C.}_{\text{Chroma Filter}}\\
&\quad=\text{C.C.}_{\text{Mslice/content-to-Bpixel}}\\
&\qquad+\text{C.C.}_{\text{generic}}+\text{C.C.}_{\text{Bpixel-to-Mslice}},\\
&\qquad\qquad\text{where C.C.}_{\text{generic}}\\
&\quad=32\times4+16\times4=192 \qquad (2)
\end{aligned}$$

Based on the proposed hybrid filtering schedule, the overall cycles are 243 and close to a lower bound of processing cycles. Table V shows a detailed cycle analysis. In our design, the neighboring pixel can be fetched from the slice memory, and the filtered results are written into the external memory without going through the slice memory. As a result, the cycle counts of the pre-process and post-process can be eliminated. In the filter-process stage, the evaluated cycle counts are 148 cycles for luma MB and 88 cycles for chroma MB. Specifically, we take 8 cycles (*DF-MB-U+ DF-MB-L*) in the $M_{\text{slice/content}}$ -to-B_{pixel} stage. There are 4×32 cycles required to filter horizontal and vertical edges in a luma MB. Moreover, we need 12 cycles (i.e., B_{pixel}-to-M_{slice}) to write the filtered results for the edges $\{16,30,32\}$. Overall, we need 148 (i.e., $8+4\times32+12$) cycles to accomplish filtering processes in a luma MB. By the same analysis, we need 88 (i.e., $4+4\times8+8=44$ for each chroma) cycles in a chroma MB. Therefore, there are total 243 cycles with extra 7 cycles for the data hazard (ie., $148+88+7$). The cycle overheads in the control logic can be neglected since it acts as a pipelined fashion. In addition, the processing cycles of the post-loop filter are identical to that of the in-loop filter because they share the same architecture and control flows. In

TABLE VI
HARDWARE SUMMARY FOR THE DEBLOCKING FILTER

Cycle Counts	[4]	[6]	[5]	Proposed
Function	in-loop	in-loop	in-loop	in/post-loop
Kept Data Size	2×4×4 sub-blocks	8×4×4 sub-blocks	2×4×4 sub-blocks	4×4×4 sub-blocks
Memory Organization	1) 96×32 + 64×32 2) External MEM	1) 96×32×2 + 64×32 2) External MEM	1) 96×32 2) External MEM	1) 96×32×2 2) 2(N+12)×32* 3) External MEM
Processing Cycles	878	446	360+N/A	243
Process	0.25μm	0.25μm	0.18μm	0.18μm
Gate Counts	20.66K	24K	11.8K	21.1K (in/post)
Clock Rate	100MHz	100MHz	100MHz	100MHz
Filtering Throughput	1.6×10^5 MB/sec	2.2×10^5 MB/sec	2.6×10^5 MB/sec	4×10^5 MB/sec

Throughput: Macro-Block/sec = $\frac{\text{Clock Rate}}{\text{Processing Cycles}}$

*__N__: frame width

conclusion, 243 cycles are close to a lower bound (192 cycles) by the proposed schedule.

To enhance the system performance, this VLSI solution is designed to achieve high throughput as well as integration efficiency. The proposal is implemented using a 0.18-μm CMOS process. Excluding the internal memory, the synthesized gate counts are 21.1 K which is reduced to 70% of the original design that realize in-loop or post-loop filtering process separately. Moreover, it achieves 4×10^5 MB/s of throughput rates when operating at 100 MHz. Finally, Table VI reveals that the throughput rates of the proposed design are about 1.5–2.5 times larger than that of existing approaches [4]–[6].

V. CONCLUSION

In this paper, the algorithms of an in/post-loop deblocking filter and its architecture have been presented. Firstly, we develop an in/post-loop deblocking algorithm that can be reconfigured as a filter for the H.264/AVC or MPEG-4 standard requirements. In particular, we propose a triple-mode decision and triple P-i-P-o edge filter to improve the integration efficiency. The overall cost can be reduced by 30% compared to the separate design. Secondly, we propose the hybrid filtering schedule to reuse the intermediate data and reduce processing cycles. We use four pixel buffers to perform the horizontal and vertical edge filter in a hybrid scheduling flow. Finally, an in/post-loop deblocking filter with hybrid filtering schedule is implemented using a 0.18-μm CMOS process. 4×10^5 MB/s of throughput rates can be achieved at a 100-MHz clock rate and is 1.5–2.5 times higher than that of existing designs [4]–[6]. Therefore, the proposal is suitable for high-throughput or multiple standard requirements such as Digital-TV and HD-DVD devices.

REFERENCES

[1] S. Srinivasan, "Windows Media Video 9: Overview and applications," *Signal Process.: Image Commun.*, vol. 19, pp. 851–875, 2004.
[2] S. D. Kim, J. Yi, H. M. Kim, and J. B. Ra, "A deblocking filter with two separate modes in block-based video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, no. 1, pp. 156–160, Feb. 1999.
[3] P. List, A. Joch, J. Lainema, G. Bjontegaard, and M. Karczewicz, "Adaptive deblocking filter," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 614–619, Jul. 2003.
[4] Y.-W. Huang, "Architecture design for deblocking filter in H.264/JVT/AVC," in *Proc. ICME*, Jul. 2003, vol. 1, p. 1-693-6.
[5] S.-C. Chang, W.-H. Peng, S.-H. Wang, and T. Chiang, "A platform based bus-interleaved architecture for de-blocking filter in H.264/MPEG-4 AVC," *IEEE Trans. Consum. Electron.*, vol. 51, no. 1, pp. 249–255, Feb. 2005.
[6] B. Sheng, W. Gao, and D. Wu, "An implemented architecture of deblocking filter for H.264/AVC," in *Proc. IEEE ICIP*, Oct. 2004, vol. 1, pp. 665–668.
[7] S. Srinivasan, T. W. Holcomb, and P. Hsu, "In-loop deblocking filter," U.S. Pat. 2005/0013494 A1, Jan. 20, 2005.
[8] T.-M. Liu, W.-P. Lee, T.-A. Lin, and C.-Y. Lee, "A memory-efficient deblocking filter for H.264/AVC video coding," in *Proc. IEEE ISCAS*, May 2005, pp. 2140–2143.