A Pipelined Hardware Architecture of an H.264 Deblocking Filter with an Efficient Data Distribution

Sang-Heon Lee and Hyuk-Jae Lee

Abstract—**In order to reduce blocking artifacts and improve compression efficiency, H.264/AVC standard employs an adaptive in-loop deblocking filter. This paper proposes a new hardware architecture of the deblocking filter that employs a four-stage pipelined structure with an efficient data distribution. The proposed architecture allows a simultaneous supply of eight data samples to fully utilize the pipelined filter in both horizontal and vertical filterings. This paper also presents a new filtering order and data reuse scheme between consecutive macroblock filterings to reduce the communication for external memory access. The number of required cycles for filtering one macroblock (MB) is 357 cycles when the proposed filter uses dual port SRAMs. This execution speed is only 41.3% of that of the fastest previous work.**

Index Terms—Block-based coding, deblocking filter, in-loop filter, H.264/AVC

I. INTRODUCTION

H.264/AVC is the newest international video coding standard approved on March 2005 by ITU-T as Recommendation H.264 and by ISO/IEC as International Standard 14496-10 (MPEG-4 part 10) Advanced Video Coding (AVC) [1]. H.264/AVC has improved significant rate-distortion performance by many useful techniques such as variable block-size motion compensation, directional spatial intra prediction, small block size discrete cosine transform, context-adaptive entropy coding, adaptive in-loop deblocking filter and etc. [2]. Among these techniques, deblocking filtering is one important tool to enhance coding efficiency. It is included inside the motion-compensated prediction path resulting in a smaller residual than previous standards. It also improves decoded video quality by smoothing artifacts caused by block-based compression. To avoid the smoothing of a true edge at a block boundary and preserve image sharpness, the deblocking filter distinguishes between a true edge and a blocking artifact. The derivations of the conditions and strengths as well as filtering operations require a large amount of computation, approximately one-third of the computational complexity of a H.264 decoder [3].

Various techniques have been proposed to implement the adaptive in-loop deblocking filter. Software implementations optimized for a general purpose processor as well as a very long instruction word (VLIW) have been proposed [4-6]. To further speed-up the execution time, a hardware implementation of deblocking filter has been widely studied [7-12]. In [7], Huang *et al.* propose an architecture that uses a reconfigurable data path to support both horizontal and vertical filterings with the same circuits. The architecture, however, has a drawback because vertical filtering is slower than horizontal filtering. Wang *et al.* [8] design a deblocking filter that includes a hardware accelerator for boundary strength (Bs). This architecture requires extra cycles for calculating the boundary strength. In [9], Sima *et al.* implement a hardwired frame-based filter. Because of the frame-based filtering, it requires a frame buffer and a longer system latency. Sheng *et al.* [10] propose a hardware architecture with 2-D filtering order to reduce the required filtering cycle. The architecture requires

Manuscript received Sep 9, 2006; revised Nov. 12, 2006.
Seoul National Univ. #054 Kwanak P.O. Box 34, Seoul 151-600, Korea
E-mail : mks@kookmin.ac.kr

extra memory to store unfiltered sample data and filtered sample data separately.

This paper proposes a new four-stage pipelined architecture to implement an adaptive in-loop deblocking filter. This architecture allows vertical filtering to be executed as fast as horizontal filtering. To feed data fast enough to fully utilize the four-stage pipeline, a new memory organization with an efficient data storage pattern is proposed to enable the access of eight samples in parallel for both horizontal and vertical filterings. This paper also presents a new filtering order and data reuse scheme to reduce the communication time for external memory access.

The rest of this paper is organized as follows. Section II presents the proposed architecture for deblocking filtering. A new memory organization with an efficient data storage pattern and the filtering order for communication time reduction are presented in Section III. Section IV evaluates the efficiency of the proposed filter and Section V presents conclusions.

II. Deblocking Filter Design

Fig. 1 shows the block diagram of the deblocking filter presented in this paper. The filter is designed to process one 16x16 macroblock (MB) and the necessary data to process one MB are transmitted from an off-chip memory to on-chip SRAM modules via an AHB bus and a memory wrapper. Control information is sent from a processor via an APB bus and stored in internal registers. The control information includes the start and end commands from the processor as well as the information necessary to calculate the boundary strength. The deblocking filter accepts eight unfiltered samples

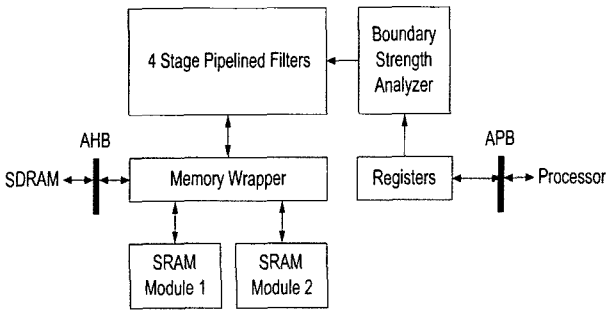


Fig. 1. Block diagram of the proposed deblocking filter.

simultaneously in one cycle and requires four cycles to process the eight samples either for a horizontal boundary or a vertical boundary. The filter is pipelined in four stages so that it can accept new eight samples at every cycle. The boundary strength analyzer calculates the Bs which is used by the pipelined filter. Since it needs two cycles for the boundary strength analyzer to calculate Bs, the filter pipeline is designed in such a way that the Bs is used at the third cycle of the pipeline resulting in the elimination of any extra cycles for the calculation of Bs.

Each of the two SRAM modules is designed to supply four samples at one cycle. Data is stored in a skewed manner to allow the supply of four samples at one cycle for both horizontal and vertical filterings. More details on the memory organization and data storage pattern are presented in the next section.

III. Data Access Pattern

1. Memory Organization and Data Storage Pattern

Fig. 2 shows the 4x4 blocks necessary to process deblocking filtering of one 16x16 MB and how these blocks are stored in memory. The Y (luma) blocks are shown in Fig. 2 (a). Each square stands for a 4x4 block and the squares numbered from one to sixteen represent the sixteen blocks that constitute one MB to be filtered. The squares labeled from A to H stand for the four upper neighboring 4x4 blocks and four left neighboring 4x4 blocks. These blocks are filtered together with the sixteen blocks. Fig. 2 (b) and (c) show Cb and Cr (chroma) blocks, respectively. One MB includes 8x8 Cb and Cr data represented by the squares numbered from seventeen to twenty four. The neighboring chroma blocks labeled from I to P are also necessary. Fig. 2 (d) shows how the blocks in Fig. 2 (a), (b) and (c) are stored in memory. The memory system is composed of two SRAM modules and each module is composed of four 8x80-bit (i.e. 1x80-byte) SRAMs. In the Fig., the leftmost memory location is addressed 0 and the address is increased from left to right. Note that neighboring blocks are stored in different memory modules in both horizontal and vertical directions so that eight samples necessary for one filtering operation can be supplied in

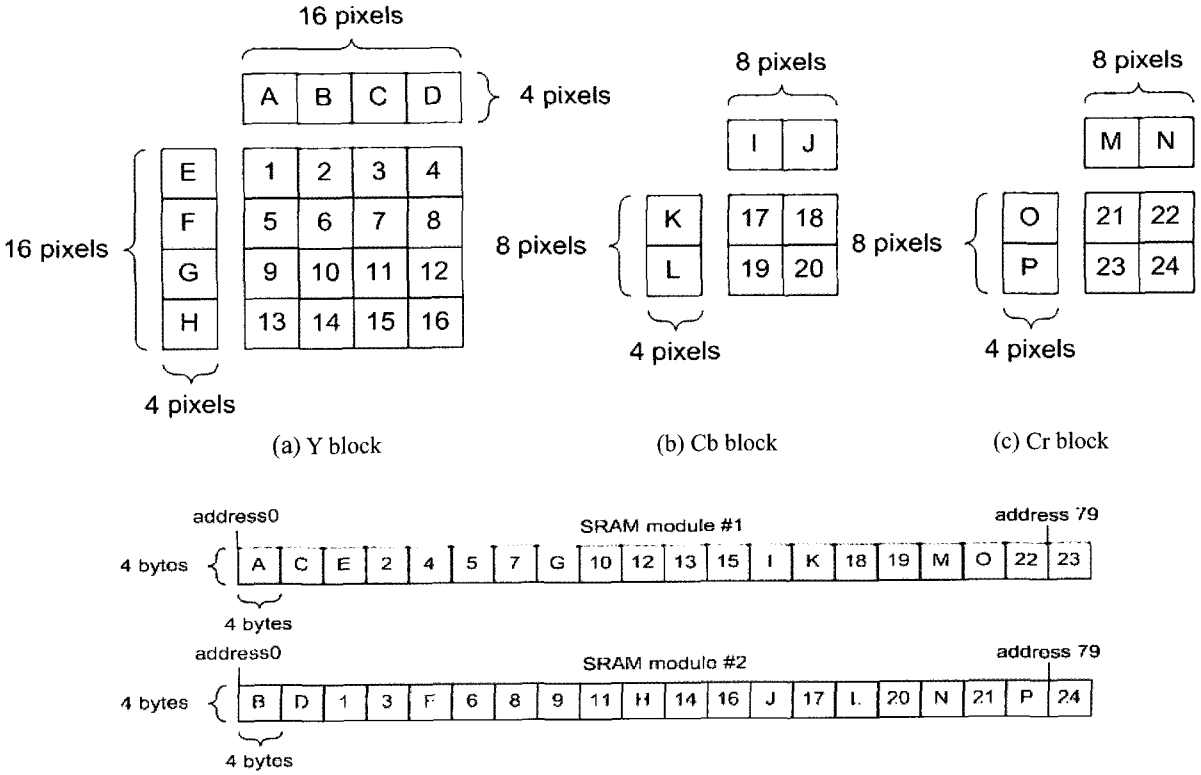


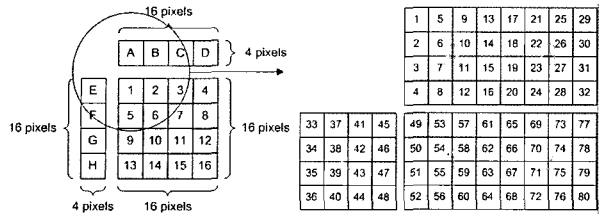
Fig. 2. Block distribution.

parallel.

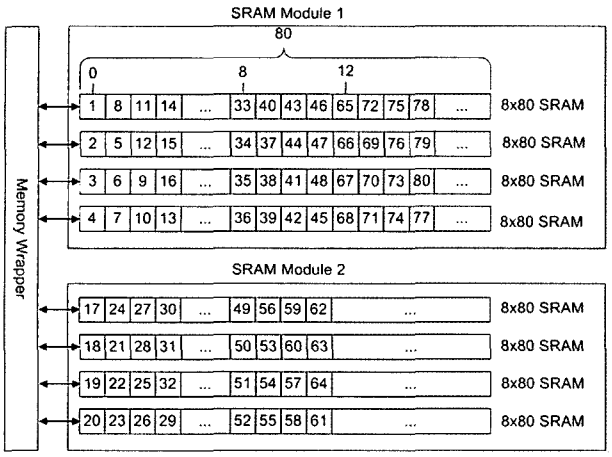
Fig. 3 shows the storage pattern of individual samples. Fig. 3 (a) shows the samples which constitute the five upper left blocks of Y data in Fig. 2 (a). Each sample is represented by a square numbered from one to eighty. Fig. 3 (b) shows the storage pattern of the samples. The memory is composed of two SRAM modules and each SRAM module is composed of four 80-byte SRAMs. One square in the Fig. represents a storage location of one byte that corresponds to one sample. The leftmost location is addressed 0 and the address is increased from left to right. Samples 1, 2, 3 and 4 are stored in the four bytes at address 0 location of SRAM module 1. Samples 8, 5, 6 and 7 are stored in the address 1 location of SRAM module 1. Note that the storing order is not 5, 6, 7 and 8 but 8, 5, 6 and 7. Similarly, the four storage elements at address 2 store four samples in the order of 11, 12, 9 and 10. The reason for the change of the storing order is to simultaneously access eight samples necessary for both horizontal and vertical filterings. More detailed explanations are described in the next paragraph. The first block consisting of samples from 1 to 16 is stored in the locations from address 0 to address 3 in SRAM module 1. The second block consisting of samples from 17 to 32 is stored in the locations addressed from 0 to 3 of SRAM module 2. All the samples of blocks A, B, E, 1 and 2 in Fig. 3 (a) are shown in Fig. 3 (b). The samples in the remaining blocks are stored in the same way.

The storage pattern in Fig. 3 can efficiently support the three types of data movements required for the H.264/AVC deblocking filtering with the architecture shown in Fig. 1. The first type of data movement is a transfer from an external memory and to an on-chip SRAM module. With a 32-bit data path, four samples can be read simultaneously. Note the column major order of Fig. 3 (a) often helps to reduce the access time because the four data samples can be stored in the memory location of the same address. The four samples are written in four different locations in the SRAM modules. For example, samples 1, 2, 3 and 4 in Fig. 3 (a) are stored into the four locations addressed 0 in SRAM module 1. Since these four samples are in different SRAMs, they can be written in parallel.

The second type of data movement is to read samples from the SRAM and send them to the pipelined filter for vertical filtering. In this case, eight vertical neighboring samples are necessary. For example, samples 1,2,3,4, 49,



(a) Example Y data



(b) Data storage pattern in SRAMs

Fig. 3. Data distribution.

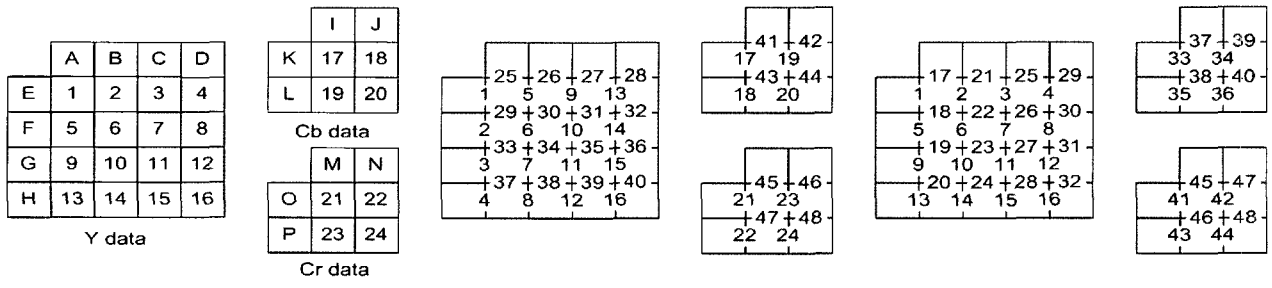
50, 51, and 52, in Fig. 3 (a) need to be read simultaneously. As shown in Fig. 3 (b), these eight samples are stored in different SRAMs so that they can be accessed simultaneously. Samples 1, 2, 3 and 4 are stored in the four SRAMs at address 0 in SRAM module 1 and samples 49, 50, 51 and 52 are stored in the four SRAMs at address 8 in SRAM module 2. As a result, these eight samples can be accessed in one cycle.

The third type of data movement occurs for horizontal filtering. Similar to the vertical filtering, eight horizontal neighboring samples are needed at one cycle for horizontal filtering. For example, samples 49, 53, 57, 61, 65, 69, 73 and 77 in Fig. 3 (a) are necessary for one horizontal filter operation. As shown in Fig. 3 (b), these eight samples are stored in different SRAMs as samples 49, 53, 57, and 61 are stored in the four different SRAMs of module 2 and samples 65, 69, 73 and 77 are also stored in the four different SRAMs of module 1. Consequently, these eight samples can be supplied to the pipelined filter at one cycle.

2. Filtering Order

This subsection presents a technique for reducing the number of memory accesses required for filtering. The result of a filtering can be used as the input data for the filtering of the next block. Therefore, the memory access for reading this input data can be eliminated. For example, consider the filtering between Blocks E and 1 in Fig. 3 (a). After the filtering is finished, the filtering between Blocks 1 and 2 is executed next. Note that the result of the filtering between Blocks E and 1 are used for the next filtering between Blocks 1 and 2. Thus, Block 1 can be fed back to the pipeline for the next filtering instead of being read again from SRAM. As a result, only Block 2 needs to be read from SRAM and a half of SRAM accesses can be eliminated.

In order to use the memory access technique discussed above, the filtering order needs to be changed. The new filtering order is explained with Fig. 4. Fig. 4 (a) shows the same Fig. again as Fig. 2 (a), (b) and (c). Fig. 4 (b) shows the filtering order required by the H.264/AVC standard. In this Fig., the filtering order is denoted by the numbers on the boundaries between blocks. The number 1 given on the block boundary between Blocks E and 1 means that the filtering for this boundary is performed first. The next filtering is performed for the block boundary between Blocks F and 5. The order of all the forty eight horizontal and vertical filterings is shown in



(a) Cb, and Cr blocks (b) H.264 standard order, and (c) Another possible order

Fig. 4. The order of filtering.

Fig. 4 (b). Note that the horizontal filterings of Y blocks are followed by the horizontal filterings of Cb and Cr blocks and then vertical filterings are executed. Fig. 4 (c) shows another possible filtering order. The order is changed such that filterings are performed in a horizontal direction first in the order while the standard order recommends filterings to be performed in the vertical order first as shown in Fig. 4 (b). This order allows the reuse of the results of the previous filtering leading to the elimination of the memory accesses of the results. Note that the filtering result is the same as the standard because the dependency of the filtering is maintained with the order and the horizontal filterings of Y blocks are followed by the vertical filterings of Y blocks. Filterings of Cb and Cr blocks are the same as those of Y blocks. The order in Fig. 4 (c) is resulting in the reduction of approximately a half of memory accesses and makes it possible to reduce required filtering cycles.

3. External Memory Access Optimization

This section investigates the time required to move data from an external memory (SDRAM) to on-chip SRAMs and proposes a new technique to reduce this communication time. Table 1 shows the pipelined execution including external memory reads and writes. In this table, 'Read' represents the stage that unfiltered samples are loaded from an external memory to an on-chip SRAM and 'Filtering' represents the stage that executes deblocking filtering. In the stage denoted by 'Write', filtered samples are read from on-chip SRAMs and stored into the off-chip memory. From the left to the right, the execution time increases and each rectangle represents one time unit (TU) which corresponds to four cycles. In 'Read' and 'Write' rows of the table, the alphabets and numbers represent the 4x4 blocks shown in Fig. 4 (a) while in 'Filtering' row, the number represents the filtering operation shown in Fig. 4 (c). 'R' represents the time required to give a new row address to the external SDRAM which is assumed to be 4 cycles in this section. 'X' represents the stalled filtering operation because data is not available or filtering results cannot be stored back to SRAMs. 'Read' begins at TU 1 with four row address strobe cycles. Then, from TU 2 to TU 6, blocks E, 1, 2, 3, and 4 are loaded from the external memory. To access the next block F, new row address strobe cycles are required because it is stored in a different row. TU 7 is used for these cycles and then blocks F, 5, 6, 7, and 8 are loaded in TU 8, 9, 10, 11, and 12, respectively. Filtering can start when the required data are stored in on-chip SRAMs. The filtering numbered 1 begins at TU 4 when the necessary blocks of E and 1 are both available. At TU 5, 6, and 7, filterings numbered 2, 3, and 4 are executed, respectively. For the filtering numbered 5, the execution unit is idle for two TUs because data are available only at TU 10. There is an idle filtering operation at TU 26 although the necessary data are available at TU 26. In this case, the short of the on-chip SRAM port causes this pipeline stall. After the filtering numbered 16 is done, the two 4x4 blocks are stored from the filter back to on-chip SRAMs. This data movement consumes TU 26. Memory write begins at TU 52 and ends at TU 102. Therefore, the total number of cycles is 408 to complete one MB filtering.

After the filtering of the current MB is finished, blocks numbered 4, 8, 12, 16, 18, 20, 22, and 24 are used for the filtering of the next MB in the right of the current MB. Therefore, the data movement for storing these blocks back to the external memory and then loading again for the next block can be removed. In this case, these blocks are stored in the locations different from the original. For example, the block numbered 4 is stored in position allocated for the block labeled E. Similarly, the blocks numbered 8, 12, 16, 18, 20, 22, and 24 are stored at the locations originally assigned for the blocks labeled F, G, H, K, L, O, and P, respectively. With the removal of data movement for these blocks, the execution time is reduced as shown in Table 2. In this case, blocks labeled E, F, G, H, K, L, O, and P are already stored and not necessary to be loaded from the external memory and blocks numbered 4, 8, 12, 16, 18, 20, 22, and 24 are not stored back to the external memory. Note that the filtering numbered 29 can only begins at TU 45 because the result of this filtering overwrite the block labeled R which, therefore, must be stored back to the external memory before the filtering numbered 29. Similarly, the blocks numbered from 30 to 48 are executed after the overwritten blocks are stored back to the external memory. From TU 80 to TU 83, filtering and memory write operations are further stalled to avoid incorrect data movements. In total, 89 TUs are required. At TU 83, the block numbered 21 is stored back to the external

Table 1. Pipelined execution of memory access and filtering.

Time Unit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	R	E	1	2	3	4	R	F	5	6	7	8	R	G	9
Filtering				1	2	3	4	X	X	5	6	7	8	X	X
Write															
Time Unit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Read	10	11	12	R	H	13	14	15	16	R	A	B	C	D	R
Filtering	9	10	11	12	X	X	13	14	15	16	X	17	18	19	20
Write															
Time Unit	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
Read	K	17	18	R	L	19	20	R	1	J	R	O	21	22	R
Filtering	X	21	22	23	24	X	25	26	27	28	X	29	30	31	32
Write															
Time Unit	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
Read	P	23	24	R	M	N									
Filtering	X	33	34	X	35	36	X	37	38	X	39	40	X	41	42
Write							R	E	1	2	3	4	R	F	5
Time Unit	61	62	63	64	65	66	67	68	69	70	712	72	73	74	75
Read															
Filtering	X	43	44	X	45	46	X	47	48						
Write	6	7	8	R	G	9	10	11	12	R	H	13	14	15	16
Time Unit	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
Read															
Filtering															
Write	R	A	B	C	D	R	K	17	18	R	L	19	20	R	I
Time Unit	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105
Read															
Filtering															
Write	J	R	O	21	22	R	P	23	24	R	M	N			

Table 2. Filtering speed and synthesized results with 4x4 block reuse.

Time Unit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	R	1	2	3	4	R	5	6	7	8	R	9	10	11	12
Filtering			1	2	3	4	X	5	6	7	8	X	9	10	11
Write															
Time Unit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Read	R	13	14	15	16	R	A	B	C	D	R	17	18	R	19
Filtering	12	X	13	14	15	16	X	17	18	19	20	X	21	22	23
Write															
Time Unit	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
Read	20	R	I	J	R	21	22	R	23	24	R	M	N		
Filtering	24	X	25	26	27	28	X	X	X	X	X	X	X	X	29
Write														R	E
Time Unit	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
Read															
Filtering	X	X	X	X	30	X	X	X	X	31	X	X	X	X	32
Write	1	2	3	R	F	5	6	7	R	G	9	10	11	R	H
Time Unit	61	62	63	64	65	66	67	68	69	70	712	72	73	74	75
Read															
Filtering	X	X	33	34	X	35	36	X	37	38	X	39	40	X	41
Write	13	14	15	R	A	B	C	D	R	K	17	R	L	19	R
Time Unit	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
Read															
Filtering	42	X	43	44	X	45	46	X	47	48					
Write	I	J	R	O	X	X	R	21	R	P23	23	R	M	N	

memory. This block is processed at TU 82 and begins to be stored at TU 83. Thus, 5 cycles is required by TU 83 to move the filtering result to on-chip SRAMs and then stored back to the external memory. In total, the number of TUs is 89 and all TUs require 4 cycles except TU 83 which requires 5 cycles. Thus, the number of total execution cycles is 357 which is 87.5% of that given Table 1.

Table 3. Comparison of filtering speed and memory require ments.

Architecture	Cycles/MB	Memory Size(bits)
Proposed	357	8DP 80x8
[7]	1,347	2SP 96x32 2SP 64x32
[10]	864	2DP 96x32 1DP 64x32
[11]	1,348	TP160x32

1. DP: Dual-port SRAM with two R/W ports.
2. SP: Single-port SRAM with one R/W ports.
3. TP: Two-port SRAM with one read and one write ports

IV. EVALUATION

This section compares the number of cycles required to perform the proposed deblocking filter for one MB. As explained in the previous section, the number of required cycles is 357 to filter one 16x16 MB including data movement to/from an external memory. Table 3 compares the proposed architecture with some previous results [7] [10, 11]. For fair comparisons, the execution cycles of the previous results are derived based on the assumption that data stored in on-chip SRAMS needs to be moved to/from an external SDRAM that requires 4 cycles of latency to access a new row. Recall that this assumption is also made for the derivation of the execution cycle for the proposed technique. Table 3 shows that [10] achieves the fastest execution time among the previous results, but the execution time of the proposed technique is only 41.3% of that of [10]. Another contribution is the inclusion of Bs calculation in the four stage pipeline so that extra computing cycles for Bs are eliminated.

V. CONCLUSIONS

This paper presents a four-stage pipelined architecture for an H.264/AVC deblocking filter. To supply the data required by the pipelined filter, this paper proposes a new data storage pattern that allows simultaneous access of eight samples for both horizontal and vertical filterings as well as efficient data movements between on-chip SRAMs and an external SDRAM. The reuse of

data between consecutive macroblock filterings is also proposed. The proposed filter with dual port SRAMs reduces the execution speed to 41.3% when compared to the fastest previous technique.

ACKNOWLEDGMENTS

This work was supported by Ministry of Commerce, Industry and Energy, Korea under contract 10023312.

REFERENCES

[1] "Draft ITU-T recommendation and international standard of joint video specification (ITU-T Rec. H.264/ISO/IEC 14496-10 AVC,"in Joint Video Team(JVT) of ISO/IEC MPEG and ITU-T VCEG, JVT-G050, 2005.

[2] T. Wiegand and G. Sullivan, G. BjØntegaard, A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Trans. Circuits Syst. Video Technol.,* vol. 13, no. 7, pp. 560-575, July 2003.

[3] P. List, A. Joch, J. Lainema, G. Bjontegaard, and M. Karczewicz, "Adaptive deblcoking filter," *IEEE Trans. Circuits Syst. Video Technol.,* vol.13, pp. 614-619, July 2003.

[4] E. Van der Tol, E. Jasper, R. H. Gelderblom, "Mapping of H.264 Decoding on a Multiprocessor Architecture," *Proc. of SPIE,* 2003.

[5] J. Golston, "DM642 Media Processor," *Proc. of SPIE*, 2003.

[6] Dang, P. P. "An Efficient Implementation of In-loop Deblocking Filters for H.264 using VLIW Architecture and Predication," *Proc. IEEE Intl. Conf. on Consumer Electronics*, 2005.

[7] Y-W Huang, T-W Chen, B-Y Hsieh, T-C Wang, T-H Chang, and L-G Chen, "Architecture Design for Deblocking Filter in H.264/JVT/AVC," in *Proc. of ICME,* 2003.

[8] Y-Y Wang, Y-T Peng, and C-J Tsai, "VLSI Architecture Design of Motion Estimator and In-Loop Filter for MPEG-4 AVC/H.264 Encoders" in *Proc. of ISCAS,* 2004.

[9] M. Sima, Y. Zhou an dW. Zhang, "An efficient architecture for adaptive de-blocking filter of H.264/AVC," *IEEE Trans. Consumer Electronics*, vol. 50, no. 1, pp. 292-296, 2004.

[10] Bin Sheng, Wen Gao, and Di Wu, "An Implemented Architecture of Deblocking Filter for H.264/AVC," *Proc. of ICIP,* 2004.

[11] Sheng-Yu Shih, Cheng-Ru Chang, and Youn-Long Lin, "An AMBA-Compliant Deblocking Filter IP for H.264/AVC," *Proc. of ISCAS,* 2005.

[12] Yo-Han Lim, Kyeong-Yuk Min, and Jong-Wha Chong, "An Efficient Architecture of Deblocking Filter in H.264/AVC for Readl-Time Video Processing," *Proc. of ELMAR,* 2005.

Sang-Heon Lee received the B.S. and M.S. degrees in Electrical and Computer Engineering at Sung-kyunkwan University, Korea, in 1997 and 1999, respectively. In September 2002, he joined Computer Architecture and Parallel Processing Laboratory at Seoul National University, Korea, where he is now a research assistant and Ph.D. student. His current research interests are low latency and error resilient processor design.

Hyuk-Jae Lee received the B.S. and M.S. degrees in Electronics Engineering from Seoul National University, Korea, in 1987 and 1989, respectively, and the Ph.D. degree in Electrical and Computer Engineering from Purdue University at West Lafayette, Indiana, in 1996. From 1998 to 2001, he worked at the Sever and Workstation Chipset Division of Intel Corporation in Hillsboro, Oregon as a senior component design engineer. From 1996 to 1998, he was on the faculty of the Department of Computer Science of Louisiana Tech University at Ruston, Louisiana. In 2001, he joined the School of Electrical Engineering and Computer Science at Seoul National University, Korea, where he is currently working as an Associate Processor. He is a founder of Mamurian Design, Inc., a fabless SoC design house for mobile multimedia applications. His research interests are in the areas of computer architecture and SoC design for multimedia applications.