Journal of Signal Processing Systems 52, 211–229, 2008
© 2008 Springer Science + Business Media, LLC. Manufactured in the United States.
DOI: 10.1007/s11265-007-0149-3

A Fast-Deblocking Boundary-strength Based Architecture Design of Deblocking Filter in H.264/AVC Applications

CHUN-LUNG HSU AND YU-SHENG HUANG
Department of Electrical Engineering, National Dong Hwa University, Hualien, Taiwan, Republic of China

Received: 15 February 2007; Revised: 7 September 2007; Accepted: 28 September 2007

Abstract. This work presents an efficient architecture design for deblocking filter in H.264/AVC using a novel fast-deblocking boundary-strength (FDBS) technique. Based on the FDBS technique, the proposed architecture divides the deblocking process into three filtering modes, namely offset-based, standard-based and diagonal-based filtering modes, to reduce the blocking artifact and improve the video quality in H.264/AVC. The proposed architecture is designed in Verilog HDL, simulated with Quartus II and synthesized using 0.18 μm CMOS cells library with the Synopsys Design Compiler. Simulation results demonstrate good performance in PSNR improvement and bit-rate reduction. Additionally, verification results through physical chip design reveal that the proposed architecture design can support 1,280×720@30 Hz processing throughput while clocking at 100 MHz. Comparisons with other studies show the excellent properties of the proposed architecture in terms of gate count, memory size and clock-cycle/macroblock.

Keywords: deblocking filter, H.264/AVC, FDBS, PSNR, bit-rate

1. Introduction

Digital video technology significantly impacts playing an important role in daily life. Block-based video coding cooperating with block transform and block motion compensation is the most extensively adopted method to decrease the data redundancy in various video coding standards, of which H.264/AVC is one of the most important and popular [1–4]. A video coding standard attempts to reduce the data correlations for achieving higher compression efficiency by employing block transform and block motion compensation approaches. However, the blocking artifact is a serious problem in many standards. To remove the blocking artifact and also enhance the coding performance, the deblocking filter acts a significant component of H.264/AVC to reduce the blocking artifact and improve the video quality.

Deblocking filter can be performed by either a post-loop or an in-loop filter. In the case of post-loop filtering, the filter is adopted outside the coding and decoding loops [5, 6]. The decoded data is stored in a data buffer, filtered and then stored in another video buffer before being forwarded to the display device. However, the in-loop filter is placed inside the loop [7–19]. The filtered frames are applied as reference frames in the motion compensation process for subsequently coded frames. The in-loop filtering is performed on macroblocks (MBs), and the filtered data can be stored directly in the video buffer to be passed onto a display device. As part of the standardization process for H.264/AVC, tests were per-

INTEL-1039
9,641,841

formed to obtain the better of these two methods. However, the deblocking filter in H.264/AVC is both computationally expensive and memory intensive owing to its highly adaptive mode decision and small 4×4 block size. The adaptive mode decision is needed for the boundary edges of the 4×4 blocks to distinguish real edges from blocking artifacts. Restated, a major challenge in designing blocking artifact detection is the detection of true edges in an image. Many studies have developed valuable approaches to eliminate the problem [5–19]. However, the deblocking operation still occupies one third of the computational complexity of a decoder, even with a highly optimized filtering algorithm. Hence, VLSI implementation is required for a highly efficient deblocking filter design for the consumer electronics market.

This work develops an efficient deblocking filter implementation based on a novel fast-deblocking boundary-strength (FDBS) algorithm to reduce the blocking artifact and enhance the video quality for H.264/AVC applications. The remainder of this paper is organized as follows. Section 2 provides a brief overview of the background work on the deblocking filter. Section 3 then describes the proposed deblocking filter architecture design based on the FDBS technique. Next, Section 4 shows the simulation results, physical implementation and performance comparisons. Conclusions are finally drawn in Section 5.

2. Background Overview

The deblocking filter is adopted to remove blocking artifacts, and thus creates a smooth picture. The inter-prediction module identifies a block similar to the current block from the reference frames. However, the reference block typically cannot match the current block perfectly, thus resulting in prediction error. The error is discrete cosine transformed and quantized for coding efficiency. The reconstructed block is different from the original block following the decoding process. In particular, discontinuity is likely to occur at the block edge. Therefore, the deblocking filter process is applied to decrease the discontinuity.

In the H.264/AVC standard, the adaptive deblocking filter is applied on edges of each 4×4 block in an MB [7]. The filtering on an MB is typically performed with raster scanning. In other words, the filtering sequence is along the vertical edges from left to right, followed by along the horizontal edges from top to bottom. The deblocking filter is invoked separately for the luma and chroma components. Figure 1 shows the processing order of the deblocking filter in an MB. The first process is horizontal filtering over the four vertical edges and transposition to intermediate data, followed by vertical filtering over the four horizontal edges. The major drawback of this direct approach is that intermediate data storage is as large as 16×16 MB in size. Furthermore, each data flow accesses a 4×4 block four times for original and intermediate data, resulting in a high bandwidth. Notably , the deblocking filter is conditional: its outputs depend on the boundary strength (BS) and the gradient of image samples across the boundary. The parameter, BS, is assigned as described by [10]. If one of the neighboring blocks is intra-coded, and the boundary is an MB boundary, then the strongest filter processing is applied (BS=4). If one of the neighboring blocks is intra-coded, but the boundary is not a MB boundary, then a fairly strong filter processing is applied (BS=3). If neither of the blocks is intra-coded, and at least one of them contains coded coefficients, then a medium strength is assigned (BS=2). Otherwise, if the blocks have different reference frames, different numbers of reference frames or different motion vector values, then a smooth filter is adopted (BS=1). If none of the previous conditions is satisfied, then filter processing is not evoked (BS=0). Notably, the parameter, BS, of any chroma component is the same as that of luma component on the same edge.

Among various coding tools in H.264/AVC, the in-loop deblocking filtering has the most significant impact on the visual quality improvement [20, 21]. To reduce the blocking artifact, the in-loop deblocking filter adaptively conducts the filtering along the boundaries of each 4×4 block based on the BS, the quantization parameter (QP) and the content of the block. Although the blocking artifact is eliminated, the computational complexity and memory access time are increased. Hence, this work develops an FDBS-based deblocking filter architecture design to improve the performance.

3. Proposed Architecture

The implementation of an efficient deblocking filter architecture design based on a novel FDBS algorithm

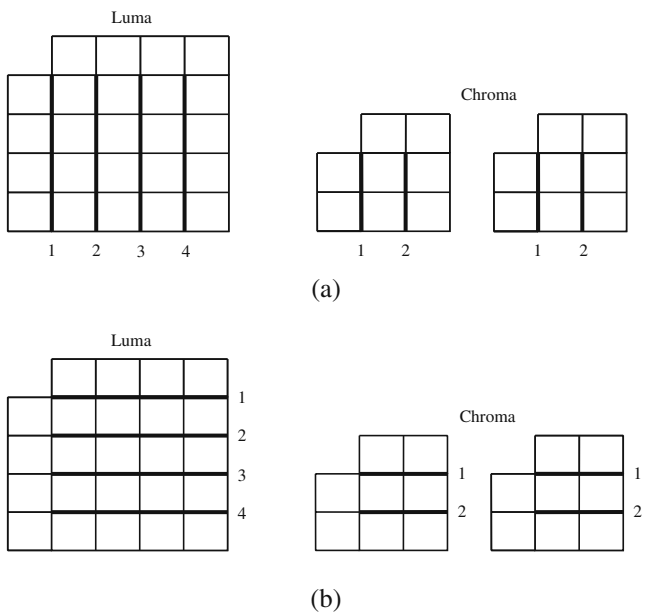


Figure 1. The order of filtering processes in an MB. **a** Horizontal filtering over vertical edges. **b** Vertical filtering over horizontal edges.

is presented here to eliminate the blocking artifacts without degrading the sharpness in the picture content for H.264/AVC applications. That is, this work develops a novel FDBS algorithm, and employs the proposed FDBS algorithm to implement the corresponding deblocking filter hardware design.

3.1. FDBS Algorithm Description

An edge filter is typically the kernel component for realizing the filtering process of the deblocking filter architecture design. Various deblocking filter algorithms for H.264/AVC have recently been presented [5–7, 16]. However, these algorithms still have complexity problems such as high computation and blocking artifact. To solve these problems, this work develops a novel FDBS algorithm to eliminate the blocking artifact, and improve the visual quality and video objective quality. Figure 2 shows the flowchart of the proposed FDBS algorithm. The proposed FDBS algorithm is applied at each 4×4 block edge of reconstructing MB, where the vertical 4×4 block edges are first filtered from left to right, and then the horizontal 4×4 block edges are filtered from top to bottom. The offset-based, standard-based and diagonal-based filtering modes are adopted to remove blocking artifacts based on the BS. Figure 2 clearly demonstrates that the offset-based filtering mode is utilized with the offset method if the BS=1. Conversely, if BS=2 or BS=3, then the standard-based filtering mode based on H.264/AVC is applied. Furthermore, the diagonal-based filtering mode is employed according to the diagonal directions (45° and 135°) when BS=4. Significantly, the value of BS affects the maximum modification of the sample values that can result from filtering. The gradation of BS reflects the strongest blocking artifacts, which are mainly caused by coding of intra-prediction error and motion compensation. The detailed analysis of the proposed FDBS algorithm is described as follows.

3.1.1. BS=1: Offset-Based Filtering Mode Figure 3 illustrates the concept of the offset-based filtering mode, in which six pixels on the top vertical and left horizontal edges of each block are filtered. The terms P_{00}, P_{01}, P_{02}, Q_{00}, Q_{01} and Q_{02} represent the pixel values after applying deblocking on pixels p_{00}, p_{01}, p_{02}, q_{00}, q_{01} and q_{02}. Notably, the difference value x is determined from the difference between two pixels across the block boundary, and importantly influences the strength of the blocking artifact. Based on the difference value, the total of

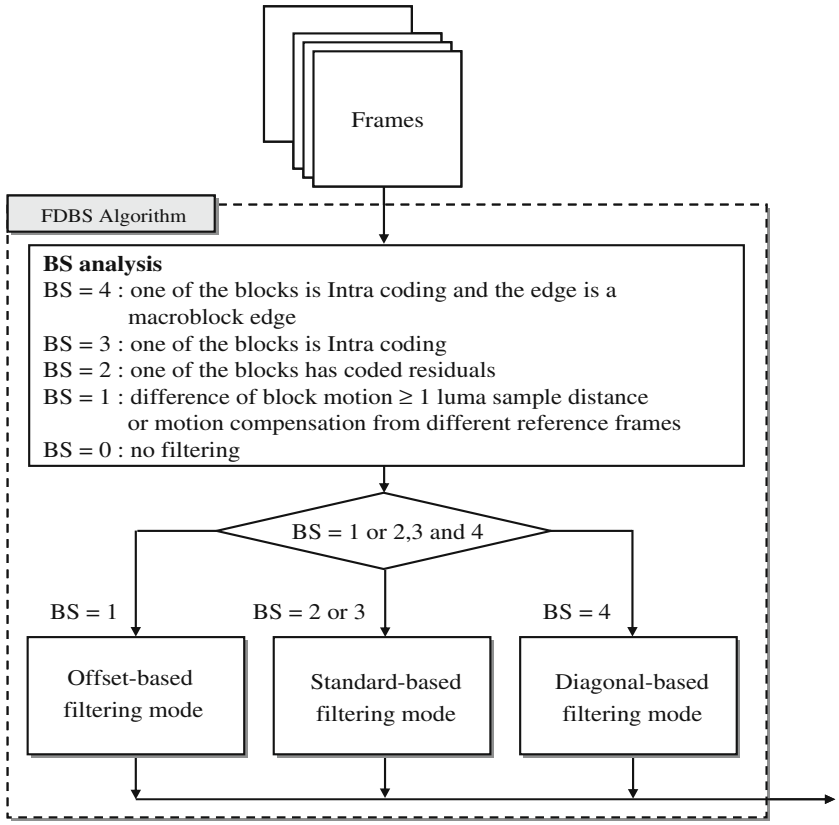


Figure 2. Flowchart of the proposed FDBS algorithm.

six pixels across the block boundary are updated and adjusted between 0 and 255. Figure 4a and b depict the corresponding one-dimensional view of the block boundaries before and after deblocking, respectively. Comparing with the H.264/AVC standard, the offset-based filtering mode removes the blocking artifacts without degrading the sharpness in the frame, and the offset-based filtering mode has

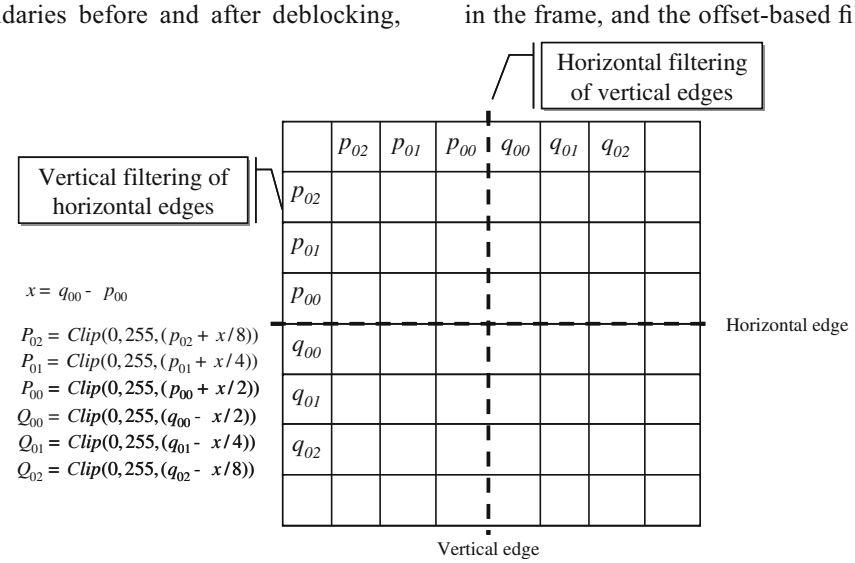


Figure 3. Description of offset-based filtering process.

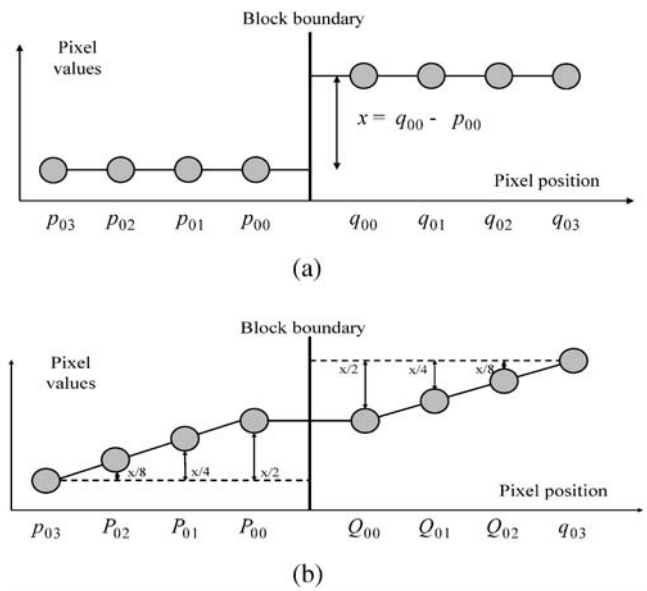


Figure 4. One dimensional view of the block boundary by offset-based filtering mode. **a** Before filtering. **b** After filtering.

extremely low computational complexity under smooth blocking artifact (BS=1). The detailed filter process of the offset-based filtering mode is discussed below.

1: $x = q_{00} - p_{00}$;
2: $P_{00} = Clip(0, 255, p_{00} + (x \gg 1))$;
3: $P_{01} = Clip(0, 255, p_{01} + (x \gg 2))$;
4: $P_{02} = Clip(0, 255, p_{02} + (x \gg 3))$;
5: $Q_{00} = Clip(0, 255, q_{00} - (x \gg 1))$;
6: $Q_{01} = Clip(0, 255, q_{01} - (x \gg 2))$;
7: $Q_{02} = Clip(0, 255, q_{02} - (x \gg 3))$;

Line 1: Determine the difference value x between two pixels.
Lines 2–4: Update p_{00}, p_{01} and p_{02}, and adjust the updated pixels to between 0 and 255.
Lines 5–7: Update q_{00}, q_{01} and q_{02}, and adjust these update pixels to between 0 and 255.

3.1.2. BS=2 or 3: Standard-based Filtering Mode. The standard-based filtering mode is enabled with the standard deblocking algorithm in H.264/AVC. Generally, a group of samples from the set (p_{02}, p_{01}, p_{00}, q_{00}, q_{01}, q_{02}) is filtered only if $BS>0$ and $|p_{00} - q_{00}| < \alpha$ && $|p_{01} - p_{00}| < \beta$ && $|q_{01} - q_{00}| < \beta$. The thresholds α and β are increased based on the average QP of the two blocks p and q. However,

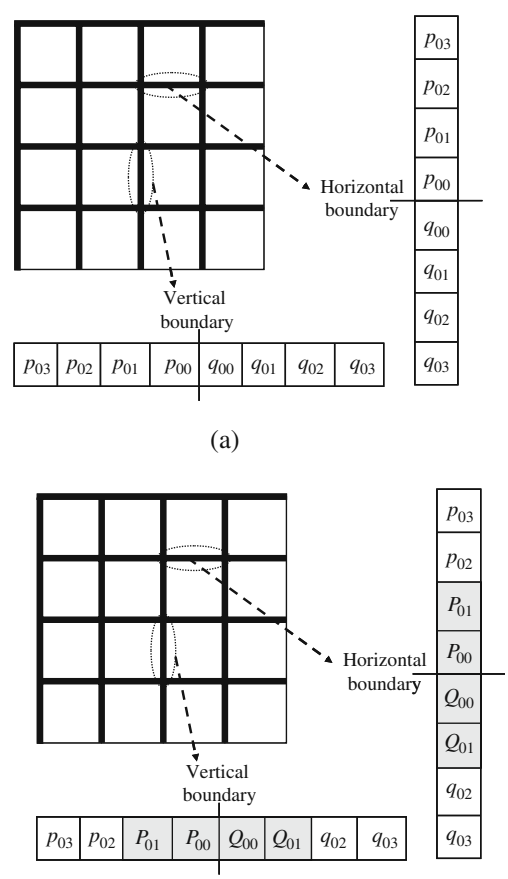


Figure 5. Pixel adjacent to vertical and horizontal boundaries by standard-based filtering mode. **a** Before filtering. **b** After filtering.

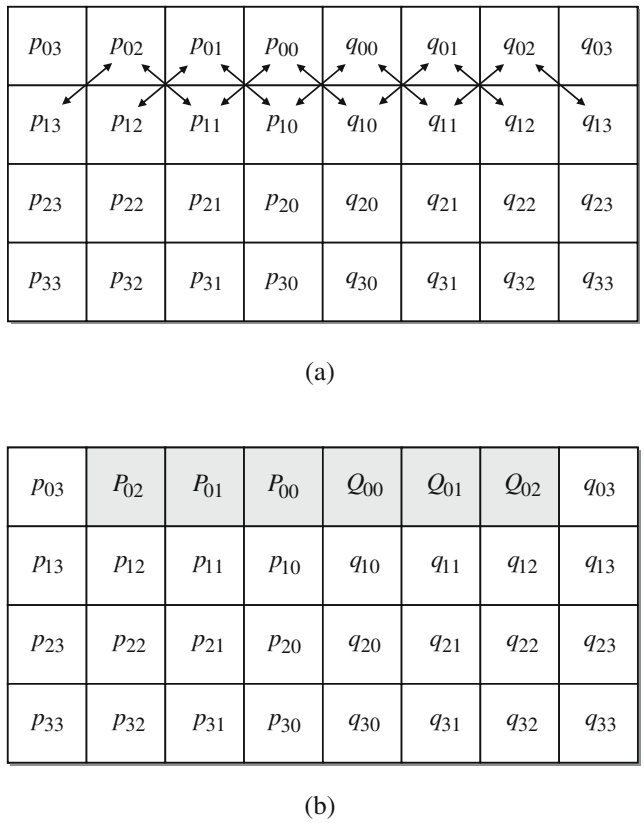


Figure 6. Diagonal-based filtering mode. **a** 45° and 135° direction condition. **b** Output of filtered process.

filtering in the horizontal and vertical directions alone only eliminate a limited number of blocking artifacts. Figure 5a illustrates four pixels on either side of a vertical or horizontal boundary in adjacent blocks p (p_{00}, p_{01}, p_{02}, p_{03}) and q (q_{00}, q_{01}, q_{02}, q_{03}). Notably, each filtering operation affects up to two pixels on either side of the boundary. In other words, either no pixels are filtered, or $p_{00}-p_{01}$ and $q_{00}-q_{01}$ are filtered to generate output pixels in the filtering process. Figure 5b shows the pixel

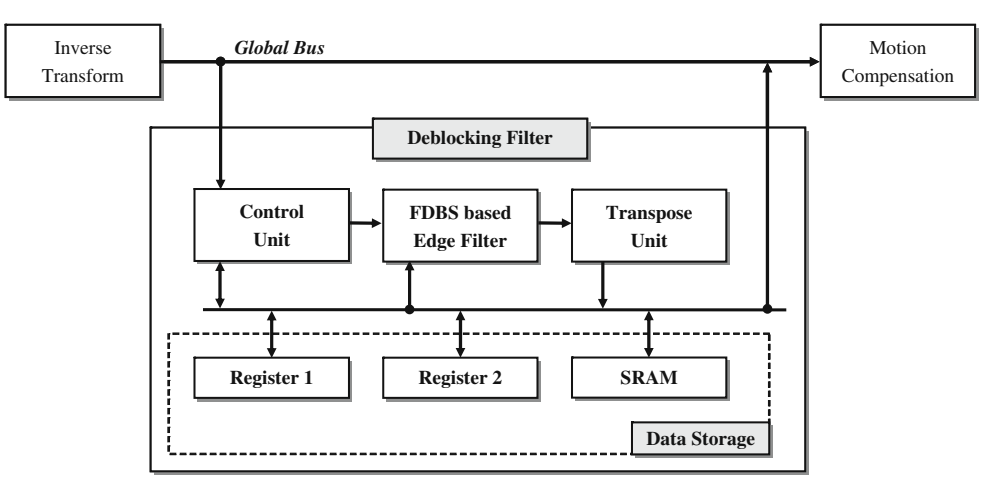


Figure 7. Proposed deblocking filter architecture design.

modification after filtering. The filtering process of the standard-based filtering mode is described as follows.

1: $\Delta = Clip(-C_0, C_0, (((q_{00} - p_{00}) \ll 2 + (p_{01} - q_{01}) + 4) \gg 3))$;
2: $P_{00} = (0, 255, (p_{00} + \Delta))$;
3: $Q_{00} = (0, 255, (q_{00} - \Delta))$;
4: *If* $|p_{02} - p_{00}| < \beta$ *then*
5: $P_{01} = p_{01} + Clip(-C_1, C_1, (p_{02} + ((p_{00} + q_{00} + 1) \gg 1) - (\mathrm{p}_{01} \ll 1)) \gg 1)$;
6: *end if*;
7: *If* $|q_{02} - q_{00}| < \beta$ *then*
8: $Q_{01} = q_{01} + Clip(-C_1, C_1, (q_{02} + ((p_{00} + q_{00} + 1) \gg 1) - (\mathrm{q}_{01} \ll 1)) \gg 1)$;
9: *end if*;

Line 1: Determine the clipping value.
Lines 2–3: Update pixels p_{00}, q_{00}.

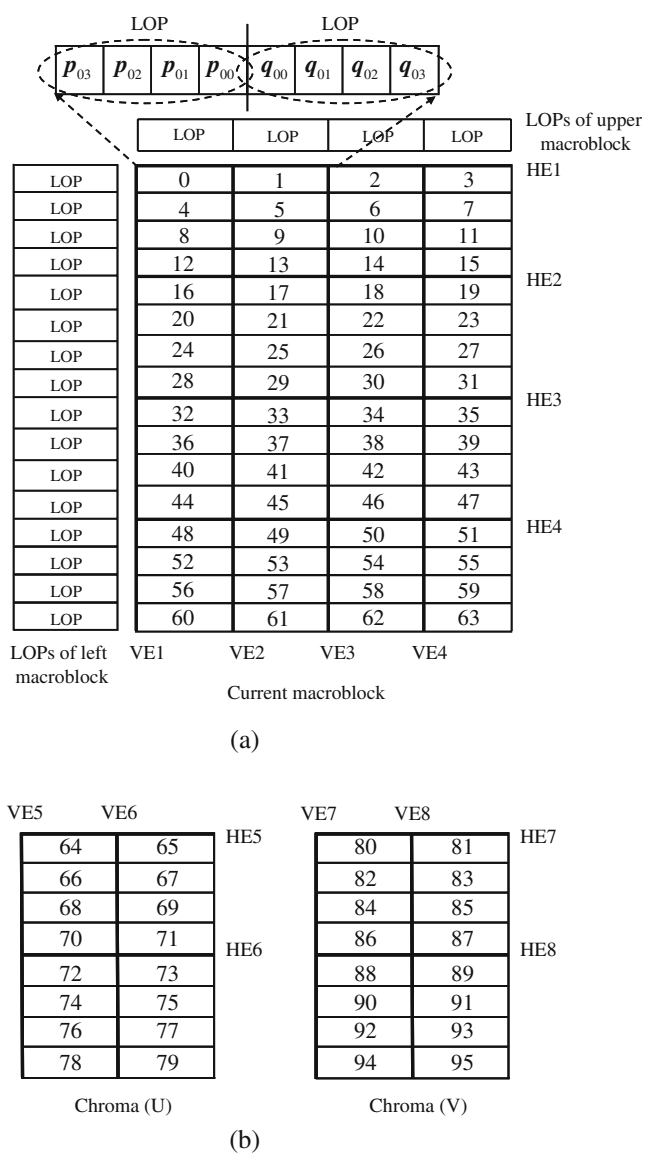


Figure 8. Data structure in the data storage unit. **a** Luma block. **b** Chroma block.

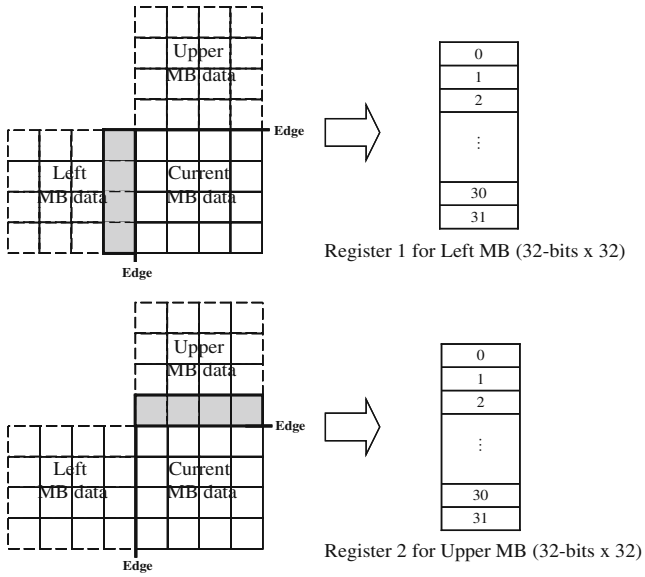


Figure 9. Register for adjacent data.

Lines 4–6: Modify sample value p_{01} to a filtered value P_{01}.

Lines 7–9: Modify sample value q_{01} to a filtered value Q_{01}.

The parameters C_0 and C_1 represent the clip variables, which are discussed in reference [2].

3.1.3. BS=4: Diagonal-based Filtering Mode. Generally, H.264/AVC will meet the problem of edge over-smoothing under serious blocking artifacts. Thus, in order to minimize the edge over-smoothing problem, we used the diagonal-based filtering mode to preclude edge pixels from deblocking filter by checking the pixel difference with a given threshold. The diagonal-based filtering mode is adopted when the diagonal directions (45° and 135°) are considered. Restated, this method removes blocking artifacts in the diagonal direction, and avoids discontinuous edge blocks. Figure 6a and b show the diagonal samples and two blocks of sample values in sides of two neighboring 4×4 blocks, respectively. Additionally, the filtering process may modify up to three pixel values on each

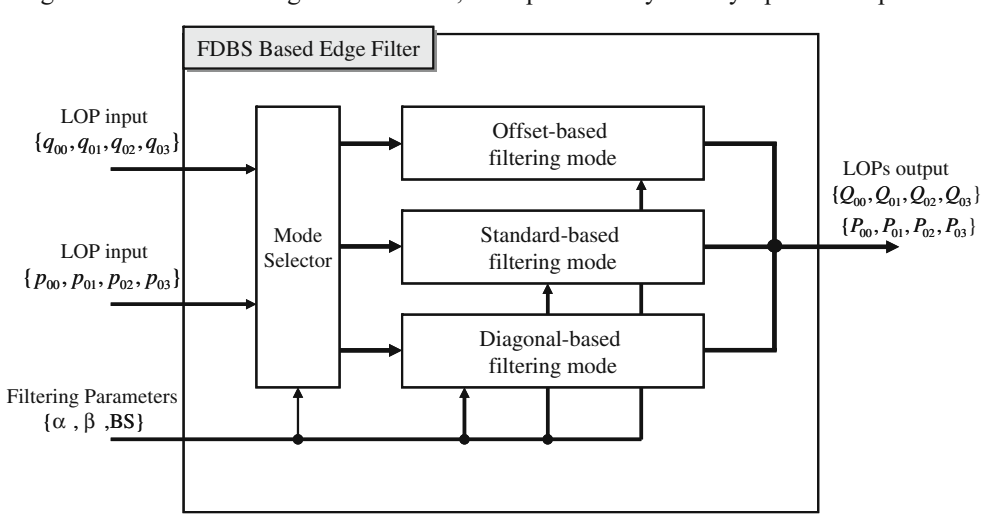


Figure 10. Block diagram of FDBS based edge filter circuit design.

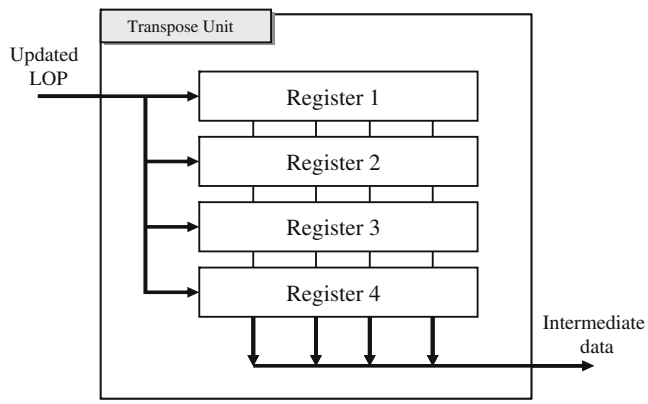


Figure 11. Block diagram of transpose unit circuit design.

side of the edge. The filtering process of the diagonal-based filtering mode is described as follows.

1 : *If* $|p_{02} - p_{00}| < \beta$ *and* $|p_{00} - q_{00}| < (\alpha \gg 2)$ *then*
2 : $P_{00} = (p_{02} + 2p_{01} + 2p_{00} + 2q_{00} + q_{01} + 4) \gg 3$;
3: *else*
4: $P_{00} = (2p_{01} + p_{00} + q_{01} + 2) \gg 2$;
5: *end if*;
6 : *If* $|p_{02} - p_{00}| < \beta$ *and* $|p_{01} - p_{12}| < \beta$ *and* $|p_{01} - p_{10}| < \beta$
7: $P_{01} = (p_{02} + p_{01} + p_{00} + q_{00} + 2) \gg 2$;
8 : *else if* $|p_{02} - p_{00}| < \beta$, $|p_{01} - p_{12}| > \beta$ *or* $|p_{01} - p_{10}| > \beta$
9: $P_{01} = (p_{00} + 2p_{01} + p_{02} + 2) \gg 2$;
10: *end if*;
11: *If* $|p_{02} - p_{00}| < \beta$ *and* $|p_{02} - p_{13}| < \beta$ *and* $|p_{02} - p_{11}| < \beta$
12: $P_{02} = (2p_{03} + 3p_{02} + p_{01} + p_{00} + q_{00} + 4) \gg 3$;
13: *else if* $|p_{02} - p_{00}| < \beta$ *and* $|p_{02} - p_{13}| > \beta$ *and* $|p_{02} - p_{11}| < \beta$
14: $P_{02} = (2p_{12} + p_{03} + 3p_{02} + p_{01} + p_{11} + 4) \gg 3$;
15: *else if* $|p_{02} - p_{00}| < \beta$ *and* $|p_{02} - p_{13}| < \beta$ *and* $|p_{02} - p_{11}| > \beta$
16: $P_{02} = (2p_{12} + p_{03} + 3p_{02} + p_{01} + p_{13} + 4) \gg 3$;
17: *end if*;
18: *If* $|q_{02} - q_{00}| < \beta$ *and* $|p_{00} - q_{00}| < (\alpha \gg 2)$ *then*
19: $Q_{00} = (q_{01} + 2p_{00} + 2q_{00} + 2q_{01} + q_{02} + 4) \gg 3$;
20: *else*
21: $Q_{00} = (2q_{01} + q_{00} + p_{01} + 2) \gg 2$;
22: *end if*
23: *If* $|q_{02} - q_{00}| < \beta$ *and* $|q_{01} - q_{10}| < \beta$ *and* $|q_{01} - q_{12}| < \beta$
24: $Q_{01} = (q_{02} + q_{01} + q_{00} + p_{00} + 2) \gg 2$;
25: *else if* $|q_{02} - q_{00}| < \beta$, $|q_{01} - q_{10}| > \beta$ *or* $|q_{01} - q_{12}| > \beta$
26: $Q_{01} = (q_{00} + 2q_{01} + q_{02} + 2) \gg 2$;
27: *end if*;
28: *If* $|q_{02} - q_{00}| < \beta$ *and* $|q_{02} - q_{11}| < \beta$ *and* $|q_{02} - q_{13}| < \beta$
29: $Q_{02} = (2q_{03} + 3q_{02} + q_{01} + q_{00} + p_{00} + 4) \gg 3$;

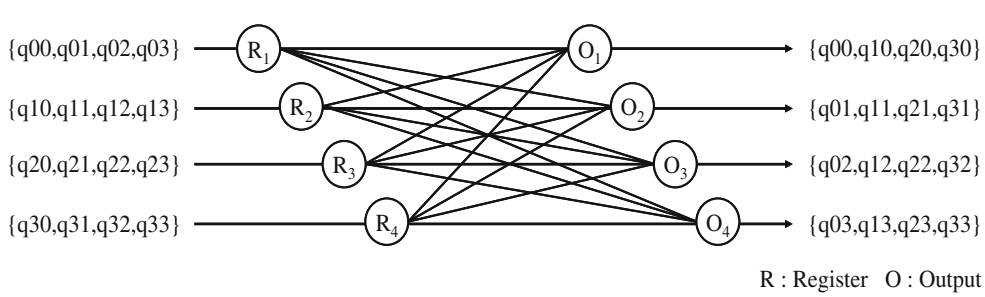


Figure 12. Data flow of transpose unit.

30: *else if* $|q_{02} - q_{00}| < \beta$ *and* $|q_{02} - q_{11}| > \beta$ *and* $|q_{02} - q_{13}| < \beta$
31: $Q_{02} = (2q_{12} + q_{03} + 3q_{02} + q_{01} + q_{13} + 4) \gg 3$;
32: *else if* $|q_{02} - q_{00}| < \beta$ *and* $|q_{02} - q_{11}| < \beta$ *and* $|q_{02} - q_{13}| > \beta$
33: $Q_{02} = (2q_{12} + q_{03} + 3q_{02} + q_{01} + q_{11} + 4) \gg 3$;
34: *end if*;

Lines 1–17: Update sample values p_{00}, p_{01} and p_{02}.
Lines 18–34: Update sample values q_{00}, q_{01} and q_{02}.

3.2. Deblocking Filter Architecture Design

Figure 7 shows the deblocking filter architecture design corresponding to the proposed FDBS algorithm. The proposed architecture is a parallel-in parallel-out design, which can be reconfigured to support different filtering strengths. Four functional blocks, namely control unit, edge filter, transpose unit and data storage, are provided to realize the functionality of the deblocking processes.

- The control unit determines the filtering parameters based on the accepted code information, and controls data input of edge filter and data storage.
- The edge filter is realized by the proposed FDBS approach, which is the kernel block of the deblocking filter.
- The transpose unit is applied to support the filtering process on vertical/horizontal edges.
- Data storage can store the sample of an MB and its adjacent blocks around the deblocking filter.

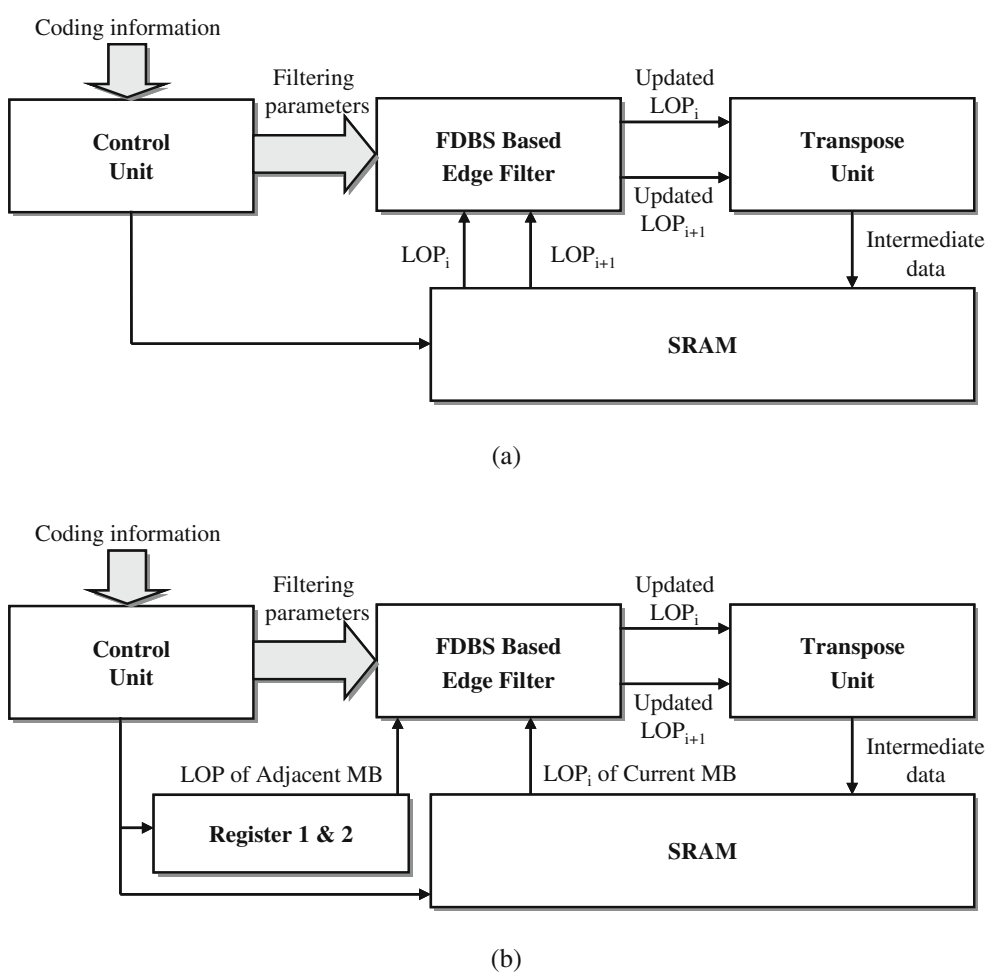


Figure 13. Data flow of horizontal filtering processes. **a** Horizontal filtering over the vertical edges VE2, VE3, VE4, VE6, and VE8. **b** Horizontal filtering over the vertical edges VE1, VE5 and VE7.

In general, the coding information from inverse transform are loaded into the control unit to determine the filter parameters α, β and BS. Based on these filtering parameters, the unfiltered data from the data storage can be filtered by the edge filter, which is designed with the proposed FDBS approach. Furthermore, the 4×4 horizontal filtering pixels on the vertical edges are stored into transpose units to transfer the horizontal filtering pixels to intermediated data for executing the 4×4 vertical filtering process on the horizontal edges. Finally, the data storage exports the filtering results to the entire architecture when the filtering processes of vertical and horizontal edges are both achieved. The details of the operation principles, related circuit designs and overall working flow are discussed as follows.

3.2.1. Data Storage Architecture Design. The data storage design of the proposed architecture comprises an SRAM and two registers. The SRAM stores the deblocking samples of a current MB, which includes one block of 16×16 luma data and two blocks of 8×8 chroma data. Figure 8 illustrates the corresponding data structure, which stores each pixel in 8 bits. Each pixel is saturated in the range 0−255 following the inverse transform. Thus, four pixels of a 4×4 block can be grouped into a 32-bit unit called the line-of-pixel (LOP). In Fig. 8, VE1-VE8 (HE1-HE8) represent the vertical (horizontal) edges to be filtered. Each square denotes an LOP, and the number in each LOP indicates the storage address of the corresponding 32-bit datum in SRAM. The 16×16

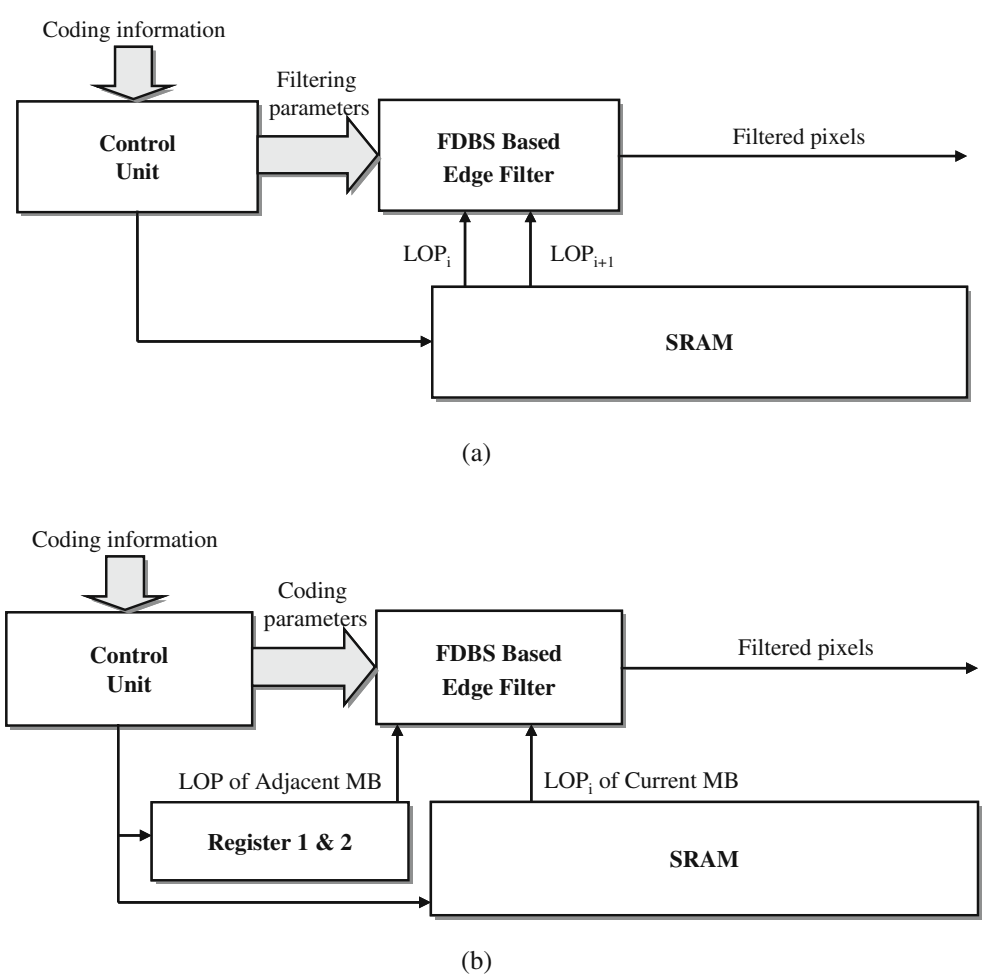


Figure 14. Data flow of vertical filtering processes. **a** Vertical filtering over the horizontal edges HE2, HE3, HE4, HE6, and HE8. **b** Vertical filtering over the horizontal edges HE1, HE5 and HE7.

luma datum in Fig. 8a is discussed as an example to explain the proposed filtering process.

– Case 1: Horizontal filtering of vertical edges

In the horizontal filtering of vertical edges, the vertical edge VE1 of the current MB is filtered first. In this case, the rightmost LOPs of the left MB and the leftmost LOPs of the current MB are needed for filtering. Notably, at most three pixels of each LOP (p_{00}–p_{02} and q_{00}–q_{02}) must be updated by the proposed FDBS algorithm. Conversely, the pixels of LOP 0 and LOP 1 are captured to update when

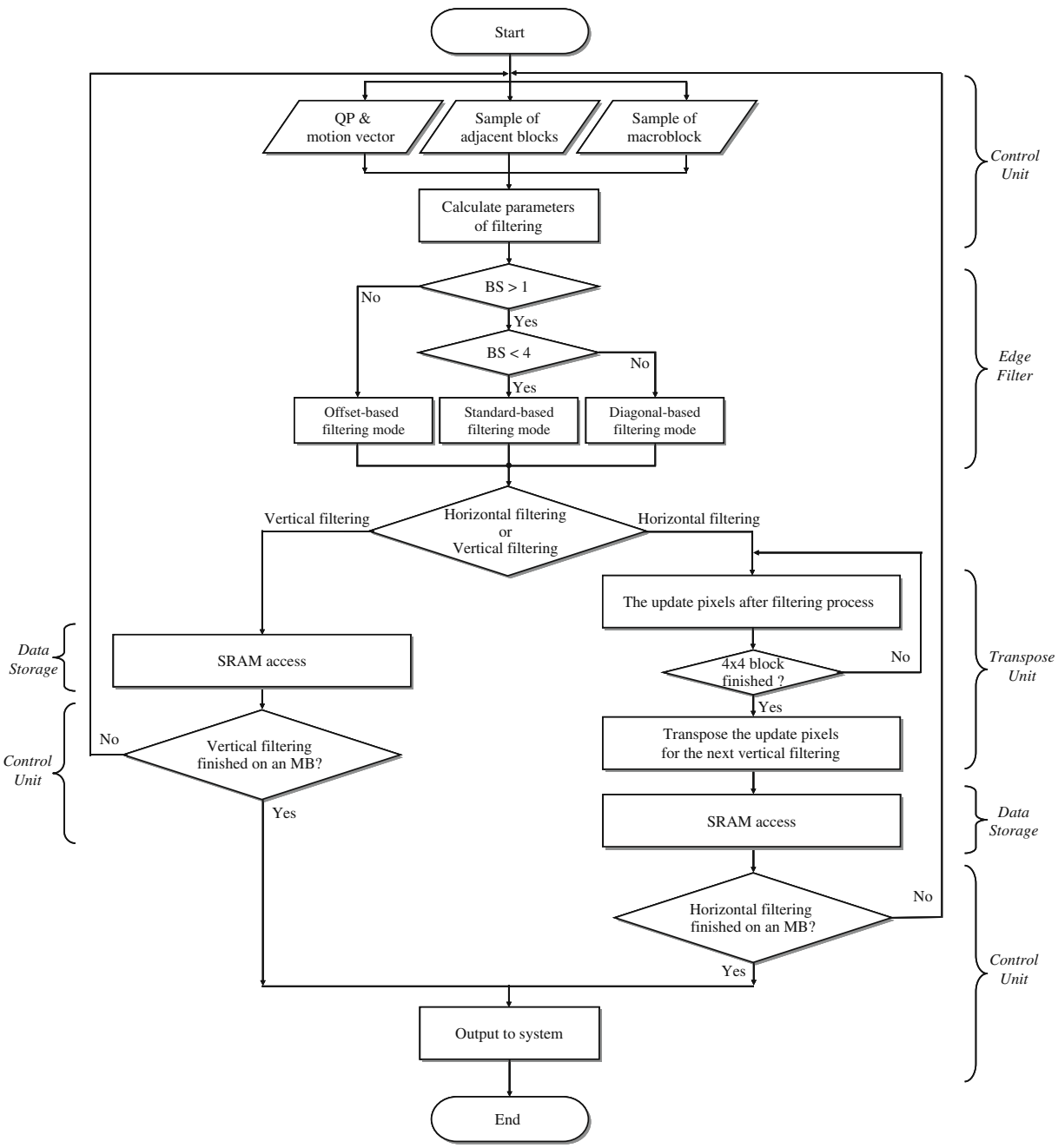


Figure 15. Overall working flow of proposed deblocking filter architecture.

the vertical edge VE2 in the current MB is considered to be filtered. All vertical edges in the current MB can be filtered using the same filtering process.

– Case 2: Vertical filtering of the horizontal edges

If the horizontal edge HE1 is regarded as filtered, then the lowest LOPs in the upper MB, and the highest LOPs in the current MB, are both needed for deblocking. Similarly, the other horizontal edges can be captured for vertical filtering using the same working principles. Restated, the LOPs are stored in the registers for the vertical filtering if the corresponding horizontal edge is considered as filtered (see Fig. 9). Additionally, the edge filter obtains the LOPs from the SRAM and registers by the proposed FDBS algorithm to replace the filtered pixels.

Furthermore, the filtering process of the chroma block edges (VE5-VE8 and HE5-HE8) of the current MB (see Fig. 8b) can be completed with the same method as described in cases 1 and 2.

3.2.2. Edge Filter Architecture Design. Figure 10 shows the block diagram of the edge filter circuit design based on the FDBS algorithm. The working processes of this design are briefly described as follows. First, two LOPs (p_{00}–p_{03} and q_{00}–q_{03}) on either side of the filtered edge from SRAM and/or registers are input to the mode selector. Meanwhile, the boundary strength parameter, BS, is also sent to the mode selector to determine the appropriate filter mode. Notably three filtering modes, i.e. offset-based, standard-based and diagonal-based, are adopted the proposed FDBS algorithm. The threshold parameters α and β are then delivered to the filter mode for filtering. Finally, the updated LOPs are sent to the transpose unit if the updated LOPs are horizontally filtered over the vertical edges. However, if the process in the edge filter is vertical filtering over the horizontal edges, then the updated LOPs are the filtered pixels, i.e., the filter process is completed.

3.2.3. Transpose Unit Architecture Design. The transpose unit is designed to temporarily store and

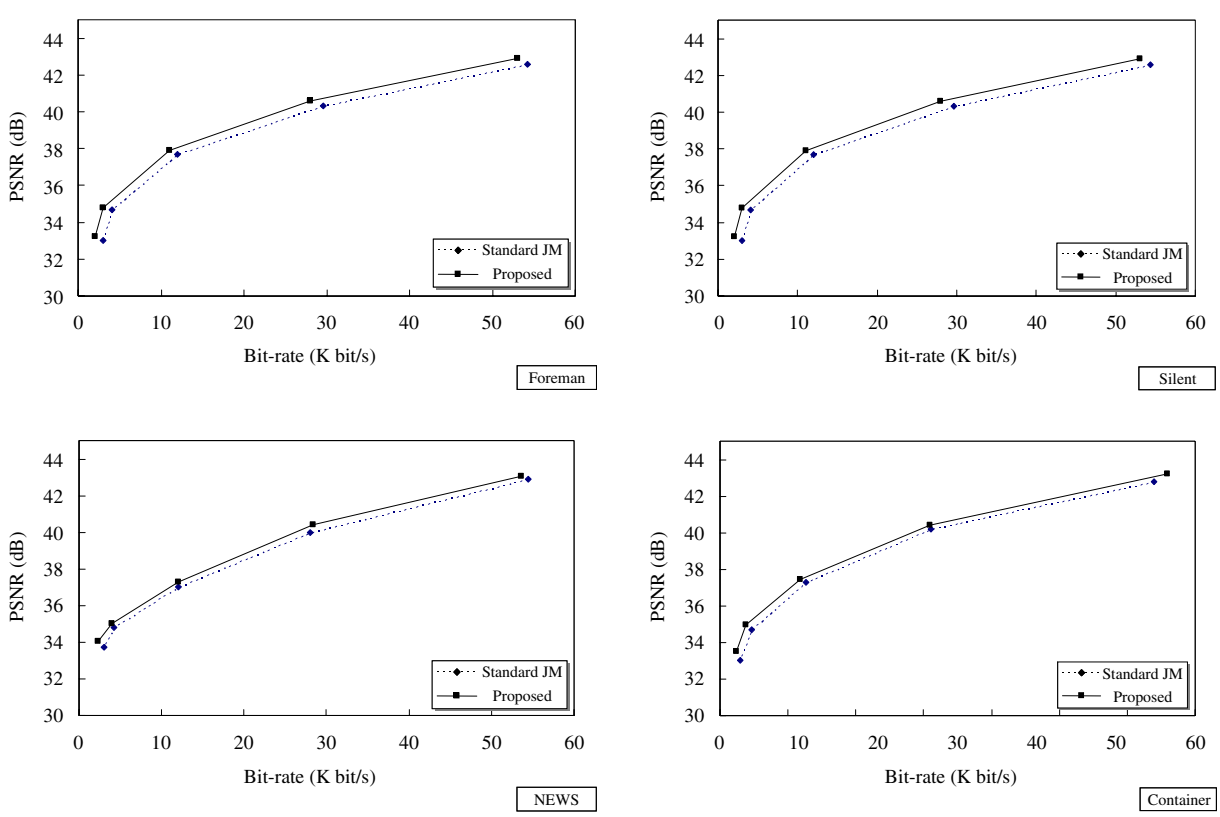


Figure 16. Simulation results of “Foreman”, “Silent”, “NEWS”, and “Container” 10 Hz QCIF chroma.

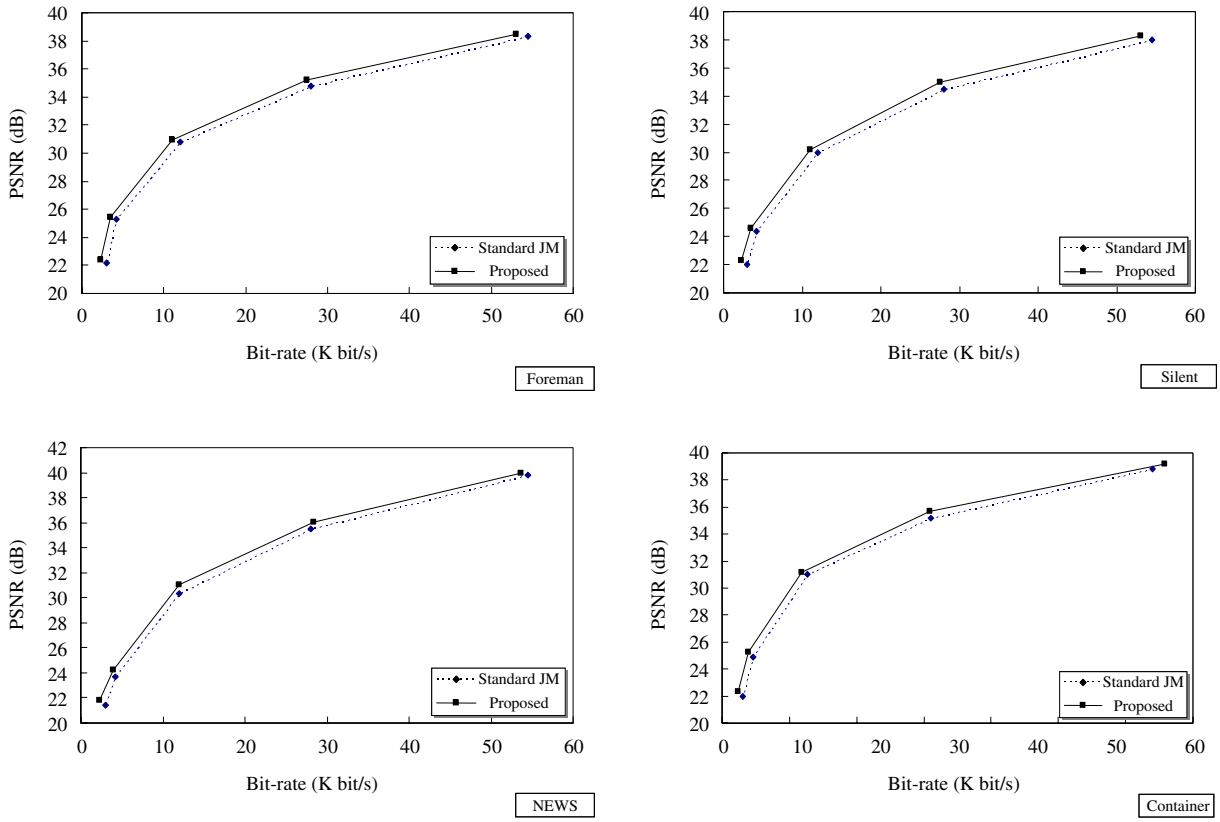


Figure 17. Simulation results of "Foreman", Silent", "NEWS", and "Container" 15 Hz QCIF luma.

transpose the pixels of the filtering 4×4 block from the edge filter. Thus, the functionality of the transpose unit in Fig. 7 is like a register. Restated, the 4×4 horizontal filtering pixels of the vertical edges are stored in the transpose unit in turn during four clock cycles, and then transposed to intermediate data between the next four clock cycles. The transpose unit thus needs eight clock cycles to complete the operations. Figures 11 and 12 show the corresponding block diagram and the data flow of the transpose unit circuit design, respectively.

Table 1. Average PNSR improvement and bit rate saving of the proposed FDBS algorithm

Sequence	PNSR improving (%)	Bit rate saving (%)
Foreman	0.34	4.1
Silent	0.35	3.6
NEWS	0.22	4.3
Container	0.24	3.0

Figures 13 and 14 illustrate the data flow to describe in detail the further operation of the proposed architecture design. Notably, the filtering process is horizontal filtering over the vertical edges, followed by vertical filtering over the horizontal edges. Figure 13a demonstrates that the coding information is loaded into the control unit to determine the filtering parameters, and transmit it to the FDBS-based edge filter. According to the filtering parameters, the FDBS-based edge filter determines the appropriate filtering mode, and grabs the unfiltered pixels (LOP_i and LOP_{i+1}) from the on-chip SRAM for horizontal filtering of the vertical edges VE2, VE3, VE4, VE6 and VE8 (see Fig. 9). The updated LOPs are then transposed to the intermediate data, and stored in the on-chip SRAM to prepare to execute the vertical filtering of horizontal edges. However, the FDBS-based edge filter also has to obtain the adjacent MB LOPs from the registers for filtering edges VE1, VE5 and VE7, since these edges are the fringe edges of the current MB. Figure 13b depicts this case. The filtering processes of the vertical

filtering of the horizontal edges are performed with a similar approach. Figure 14a and b show the data flow of the vertical filtering of horizontal edges HE2, HE3, HE4, HE6 and HE8, and HE1, HE5 and HE7, respectively. Notably, the transpose unit is not necessary in the vertical filtering processes, because the output of the FDBS-based edge filter are the filtered pixels.

Based on the discussion of each sub-circuit and the data flow analysis, the overall working flow of the proposed deblocking filter architecture design is summarized in Fig. 15, and is briefly described as follows.

Step 1: Obtaining the coding information from global bus.

The parameter QP, motion vectors, samples of current MB and adjacent MB from inverse transform are loaded into the control unit for calculation.

Step 2: Using the control unit to calculate the filtering parameters.

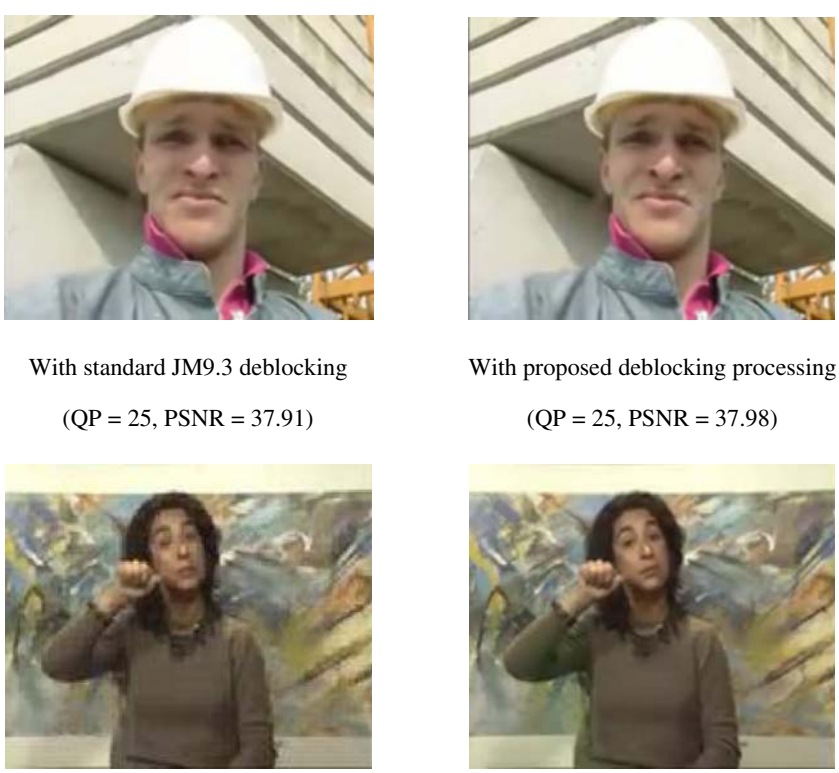
The accepted code information (QP and motion vectors) is applied to derive the filtering parameters α, β and BS.

Step 3: Filtering the vertical and horizontal edges.

The FDBS-based edge filter can obtain the unfiltered data from memory and register for filtering according to the filter parameters. For each MB, horizontal filtering of the vertical edges is performed first, followed by vertical filtering of the horizontal edges. Notably, both filtering directions must be performed on each MB before moving on to the next MB.

Step 4: Transposing filtering pixels to intermediated data.

The filtering processes of the proposed architecture is to filter the vertical 4×4 block



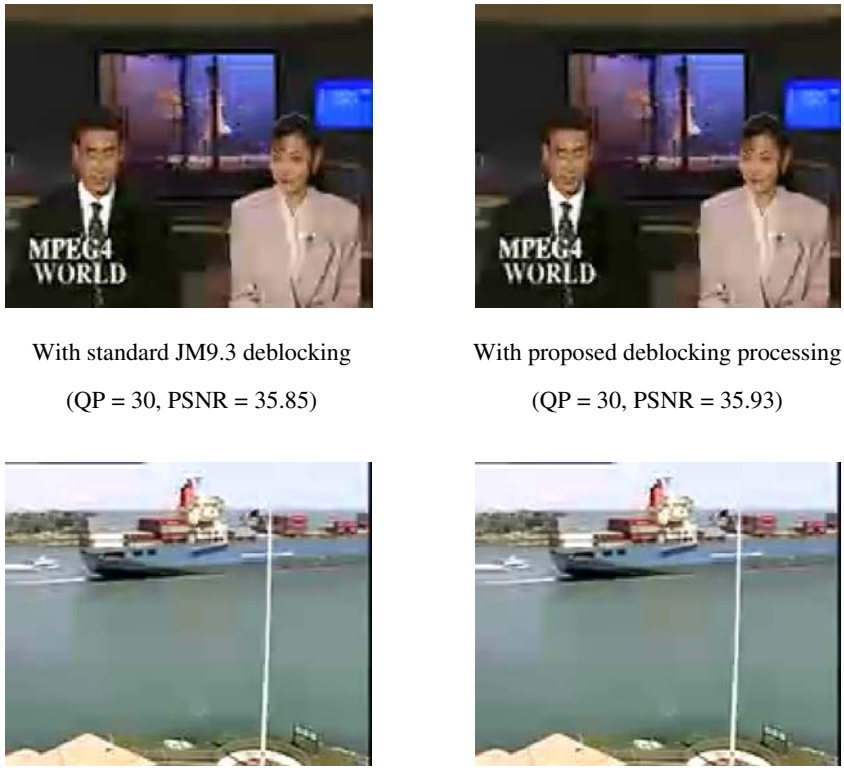
With standard JM9.3 deblocking (QP = 25, PSNR = 37.91)

With proposed deblocking processing (QP = 25, PSNR = 37.98)

With standard JM9.3 deblocking (QP = 25, PSNR = 38.35)

With proposed deblocking processing (QP = 25, PSNR = 38.43)

Figure 18. Deblocking result of “Foreman” and “Silent” video sequences.



With standard JM9.3 deblocking (QP = 30, PSNR = 35.85)

With proposed deblocking processing (QP = 30, PSNR = 35.93)

With standard JM9.3 deblocking (QP = 30, PSNR = 35.59)

With proposed deblocking processing (QP = 30, PSNR = 35.67)

Figure 19. Deblocking result of "NEWS" and "Container" video sequences.

edges from left to right first, and then deblock the horizontal 4×4 block edges from top to bottom. Thus, the 4×4 horizontal filtering pixels on the vertical edges are stored in the transpose unit in turn during four clocks, and these pixels are then transposed into intermediate data at the other four clocks in preparation for execution of the 4×4 vertical filtering on the horizontal edges.

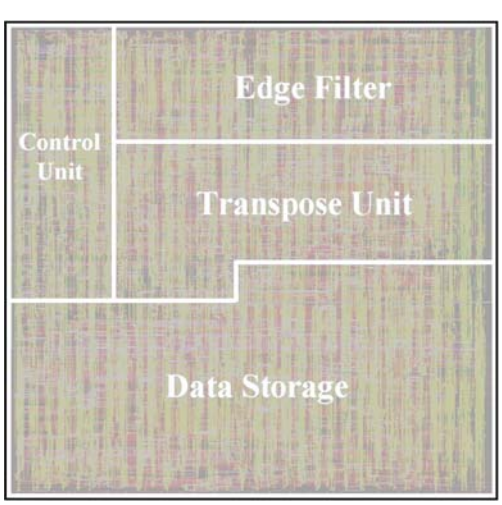


Figure 20. Physical implementation of the proposed architecture.

Step 5: Checking that the filtering process are finished.

The filtering results are exported to the entire architecture when the filtering processes of the vertical and horizontal edges are both accomplished.

4. Results Discussion

To evaluate the accuracy and efficiency of the proposed deblocking filter, the architecture was designed by Verilog HDL and implemented with the TSMC 0.18 μm 1P6M CMOS technology. The

Table 2. Comparison results

	Proposed	[10]	[15]	[19]
Technology	0.18 μm	0.25 μm	n/a	0.25 μm
Gate count (100 MHz) (without SRAM)	18.53 K	24 K	n/a	20.66 K
Cycle/MB	366	446	>600	614
Memory size (bits)	32×96	32×64×1 32×96×2	Frame size	32×160
Processing rate (100 MHz)	1,280×720@30 Hz	1,280×720 @60 Hz	1,280×720@30 Hz	1,280×720@30 Hz
Memory architecture	Two port	Dual port ×1 Two port ×2	Two port	Two pot

detailed simulation results, physical implementation and performance comparisons are discussed as follows.

4.1. Simulation Results

The proposed FDBS algorithm was implemented into H.264/AVC reference software, JM9.3 to demonstrate the compatibility of the standard JM code, and to measure the performance improvement in PSNR, bit-rate and subjective quality. The visual test was performed on the JM9.3 coded sequences, such as "Foreman", "Silent", "NEWS", and "Container". Figures 16 and 17 show the simulation results of QCIF chroma and luma, respectively. The simulation results clearly reveal that the performance of the proposed FDBS algorithm is better than standard JM code in both PSNR improvement and bit-rate reduction. The average PSNR improvement and bit rate saving of the proposed FDBS algorithm are listed in Table 1.

On the other hand, the subjective quality is another criterion for deblocking filter design. Using the standard JM and proposed FDBS algorithms, the subjective quality observation of sequences "Forman", "Silent" with QP=25 and "NEWS", "Container" with QP=30 are illustrated in Figs. 18 and 19, which demonstrate that the proposed FDBS algorithm obviously enhance the subjective views and PSNR values.

4.2. Implementation and Comparisons

The validated Verilog HDL code was also synthesized with the TSMC 0.18 μm CMOS cell library by the Synopsys Design Compiler. Figure 20 illustrates the chip layout of the proposed architecture design. The core size is about 736×573 μm^2. Table 2 lists the comparisons of synthesized results between previous works and the proposed architecture, and clearly demonstrates that the proposed architecture can support 1,280×720@30 Hz processing throughput

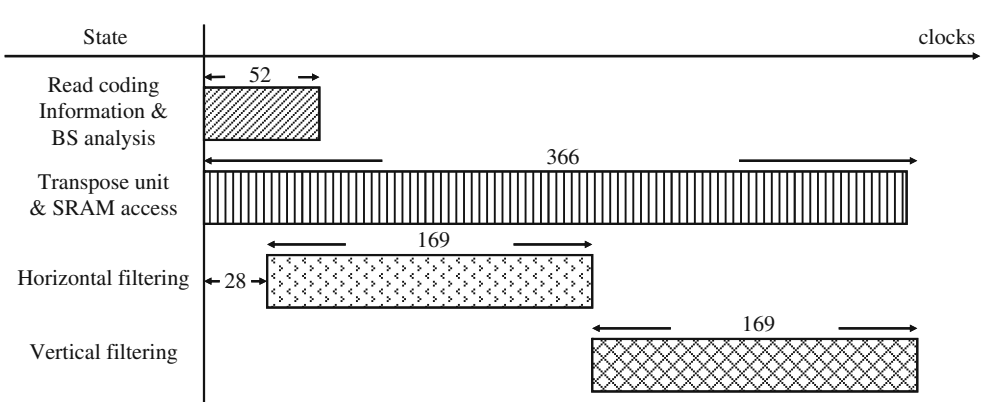


Figure 21. Processing scheduling chart of the proposed architecture.

while clocking at 100 MHz. Additionally, the proposed architecture possesses good characteristics in gate count, memory size and clock-cycle/MB. Figure 21 depicts the required processing clocks of the proposed architecture to describe the clock-cycle/MB further. According to Fig. 21, the proposed architecture takes 52 clock cycles to read the coding information and perform the BS analysis. Moreover, 338 [186 (Luma) + 152 (Chroma)] clocks are needed for both horizontal and vertical filtering. Thus, the proposed architecture performs deblocking filtering for all MBs in 366 cycles, which is adequate to realize the real-time deblocking filter of a HDTV (1,280×720 30fps) H.264/AVC bitstream.

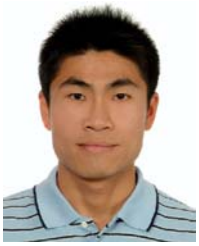
5. Conclusions

This work develops a hardware architecture design of a deblocking filter based on a novel FDBS algorithm. The FDBS algorithm adopts offset-based, standard-based and diagonal-based filtering modes to eliminate annoying blocking artifacts and improve subjective video quality. The Verilog HDL coding and TSMC 0.18 μm CMOS technology are adopted herein to realize and verify the proposed FDBS approach and the corresponding architecture design. Simulation results, physical implementation and performance comparisons demonstrate that the proposed deblocking filter architecture design can effectively achieve PSNR improvement and bit-rate saving with a small number of gate counts and clock cycles. Thus, the processing capability of the proposed architecture can operate quite efficiently for real-time applications, and are highly promising for integration into an H.264/AVC codec.

References

1. T. Wiegand, G. J. Sullivan, G. Bjntegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, Jul. 2003, pp. 560–576.
2. T. Wiegand, *Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264|ISO/IEC 14496-10 AVC),* JVT-G050, 2003.
3. J. Ostermann, J. Bornans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wdei, "Video Coding with H.264/AVC: Tools, Performance, and Complexity," *IEEE Circuits Syst. Mag.*, vol. 4, 2004, pp. 7–28.
4. S. Wenger, "H.264/AVC Over IP," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, Jul. 2003, pp. 645–656.
5. S. C. Tai, Y. Y. Chen, and S. F. Sheu, "Deblocking Filter for Low Bit Rate MPEG-4 Video," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, Jun. 2005, pp. 733–741.
6. S. C. Tai, Y. R. Chen, C. Y. Chen, and Y. H. Chen, "Low Complexity Deblocking Method for DCT Coded Video Signals," *Proc. IEE Image Signal Process.*, vol. 153, Feb. 2006, pp. 46–56.
7. P. List, A. Joch, J. Lainema, G. Bjntegaard, and M. Karczewicz, "Adaptive Deblocking Filter," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, Jul. 2003, pp. 614–619.
8. G. Zheng and L. Yu, "An Efficient Architecture Design for Deblocking Loop Filter," in *Proc. the Picture Coding Symp.*, San Francisco, California, USA, Dec. 2004.
9. S. C. Chang, W. H. Peng, S. H. Wang, and T. Chiang, "A Platform Based Bus-Interleaved Architecture for Deblocking Filter in H.264/MPEG-4 AVC," *IEEE Trans. Consum. Electron.*, vol. 51, Feb. 2005, pp. 249–255.
10. B. Sheng, W. Gao, and D. Wu, "An Implementation Architecture of Deblocking Filter for H.264/AVC," *Proc. IEEE Int. Conf. Image Processing*, vol. 1, Oct. 2004, pp. 665–668.
11. V. Venkatraman, S. Krishnan, and N. Ling, "Architecture for Deblocking Filter in H.264," in *Proc. the Picture Coding Symp.*, San Francisco, California, USA, Dec. 2004.
12. L. Lingfeng, S. Goto, and T. Ikenaga, "An Efficient Deblocking Filter Architecture with 2-dimensional Parallel Memory for H.264/AVC," *Proc. Asia S. Pac. Des. Automat. Conf.*, vol. 1, Jan. 2005, pp. 623–626.
13. S. Y. Shih, C. R. Chang, and Y. L. Lin, "A Near Optimal Deblocking Filter for H.264 Advanced Video Coding," *Proc. Asia S. Pac. Des. Automat. Conf.*, Jan. 2006, pp. 170–175.
14. C. C. Cheng, T. S. Chang, and K. B. Lee, "An In-place Architecture for the Deblocking Filter in H.264/AVC," *IEEE Trans. Circuits Syst. II: Analog Digit. Signal Process.*, vol. 53, 2006, p. 1.
15. M. Sima, Y. H. Zhou, and W. Zhang, "An Efficient Architecture for Adaptive Deblocking Filter of H.264/AVC Video Coding," *IEEE Trans. Consum. Electron.*, vol. 50, Feb. 2004, pp. 292–296.
16. Y. C. Zhu and M. J. Chen, "Improvement of Adaptive Deblocking Filter in H.264," *Workshop on Consumer Electronics*, Hsinchu Taiwan, Nov. 2004.
17. T. M. Liu, W. P. Lee, T. A. Lin, and C. Y. Lee, "A Memory-efficient Deblocking Filter for H.264/AVC Video Coding," *Proc. IEEE Int. Symp. ISCAS*, vol. 3, May 2005, pp. 2140–2143.
18. C. C. Cheng and T. S. Chang, "An Hardware Efficient Deblocking Filter for H.264/AVC," *Proc. Int. Conf. Consumer Electron. (ICCE)*, Jan. 2005, pp. 235–236.
19. 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/AVC," *Proc. IEEE Int. Conf. on Multimedia and Expo 2003*, vol. 1, Jul. 2003, pp. 693–696.
20. G. Khurana, A. A. Kassim, T. P. Chua, and B. M. Michael, "A pipelined Hardware Implementation of in-loop Deblocking Filter in H.264/AVC," *IEEE Trans. Consum. Electron.*, vol. 52, May 2006, pp. 536–540.
21. S. D. Kim, J. H. Lee, J. M. Yang, M. H. Sunyyoo, and S. K. Oh, "Novel Instructions and their Hardware Architecture for Video Signal Processing," *IEEE Int. Symp. Circuits Syst.*, vol. 4, May 2005, pp. 3323–3326.

Chun-Lung Hsu received the Ph.D. degree in Electrical Engineering from National Taiwan University, Taipei, Taiwan, R.O.C. in 2000. He is interested in the relationship of lowpower circuit design, consumer electronics development, system on a chip design and VLSI testing. He is currently working as an Associate Professor in the Department of Electrical Engineering, National Dong Hwa University, Hualien, Taiwan, R.O.C. The major research of his laboratory are video IC design, built-in self-detect/correct for video coding, and tolerance scheme development for image system. He is now a member of the IEICE, IET, and IEEE Society.



Yu-Sheng Huang was born in Taichung, Taiwan, R.O.C. in 1981. He received the Bachelor of Engineering degree in Computer Science and Information Engineering of Chaoyang University of Technology, Taichung, Taiwan, R.O.C. in 2004, and M.S. degree of Engineering in National Dong Hwa University, Hualien, Taiwan, R.O.C. in 2006. His research focused on video IC design, built-in self-test and fault/error tolerance. Huang is currently pursuing the Ph.D. in National Dong Hwa University.