Sensors and Materials, Vol. 31, No. 3 (2019) 905–922
MYU Tokyo

S & M 1824

Hardware Architecture Design for High-performance H.264/AVC Deblocking Filter

Xiaodong Zheng,[1] Wei Zhu,[1*] Shaoyong Yu,[2] and Jinpeng Wu[1]

[1]School of Software Engineering, Xiamen University of Technology,
No. 600, Ligong Road, Jimei District, Xiamen City 361024, China
[2]School of Mathematics and Information Engineering, Longyan University,
No. 1 Dongxiaobei Road, Xinluo District, Longyan City, Fujian Province 364012, China

(Received October 15, 2018; accepted January 30, 2019)

Keywords: block-based discrete cosine transform (BDCT), block effect, deblocking filter, low memory, low cost

In multimedia imaging/video compression systems, the block-based discrete cosine transform (BDCT) is widely applied to conversion and compression. During the quantization of compression, after processing with an inverse discrete cosine transform (IDCT), a certain defect emerges in BDCT, i.e., two adjacent blocks fail to be perfectly joined together. This block effect of the picture undermines the video quality and tends to become more pronounced as the sampling factor increases. In image decoding, a deblocking filter requires an enormous amount of computation. If the computation is carried out entirely by software, a large system bandwidth will inevitably be occupied. To overcome this problem, the design of a suitable hardware architecture is urgently required. In this study, we proposed the use of the system bus and a low-memory, low-cost, and effective deblocking filter architecture to reduce the operation time of the deblocking filter. In contrast to existing hardware architectures, the design proposed in this paper adopts dual-port static random access memory (SRAM) and two-port SRAM architectures, thereby storing data via a new data reading approach, to save transposed memory and reduce the hardware size. By coupling the proposed architecture with parallel processing units, the processing speed will be increased.

1. Introduction

With the booming of information technology, various multimedia technologies are often used via Internet applications, such as video conferencing, video on demand, and video surveillance. Network technology has led to the introduction of broadband internet access; however, with the increasing number of users and the higher demand for picture quality, the network bandwidth is expected to be saturated soon. Therefore, multimedia video compression technology will be heavily relied on to compress data, so as to store more multimedia data in existing storage spaces, which in turn can reduce the time required for data transformation via the Internet.

The H.264/AVC standard for video compression has recently been jointly developed

*Corresponding author: e-mail: zhw@xmut.edu.cn
https://doi.org/10.18494/SAM.2019.2163

ISSN 0914-4935

by ITU-T (Video Coding Experts Group, VCEG) and ISO/IEC (Moving Picture Experts Group, MPEG).[1] The picture quality, compression efficiency, and error tolerance of H.264/AVC significantly outperform those of previous video compression standards. Compared with other video compression standards,[2] in the case of low-data movies, H.264/AVC can provide higher picture quality and compression efficiency. However, the significant increase in the computational complexity of the signal results in many obstacles for the use of H.264/AVC in real-time systems. Similarly to other existing video compression systems, the block-based discrete cosine transform (BDCT) and quantization based on block processing are commonly used, whose signal is processed by dividing the entire picture into non-overlapping blocks. In BDCT, each block is then converted from a space domain to a frequency domain. Then, the obtained coefficient is divided by a quantization-parameter-based quantization matrix. As a result of this processing, the high-frequency signal invisible to human eyes is removed, thereby achieving data compression. However, when a relatively large quantization parameter is used, the BDCT coefficient obtained after quantization and the amount of data are sufficient, indicating that the quantization parameter directly leverages the compression quality of a video system.

A deblocking filter is involved in the next-generation video compression system based on H.264/AVC and called the in-loop filter. Traditionally, a deblocking filter is integrated into a video codec as a postfilter, then the built-in deblocking filter will process the picture into a reference image.[3] In contrast, a H.264/AV deblocking filter is equipped with a mechanism that is highly adaptable to different picture sources, which produces higher video quality.[4] In the H.264/AVC standard, the deblocking filter is an essential mechanism for ensuring video quality. Taking into account the fact that the prediction, conversion, quantization, and motion compensation blocks have minimum sizes of 4 × 4, to remove the blocking effect in the picture, the deblocking filter must filter every 4 × 4 matrix in the picture, that is, almost every pixel must be processed by the deblocking filter. To this end, it is necessary to carry out complicated reading and access of the memory storing the picture signals. In addition, H.264/AVC utilizes various thresholds and conditions to determine and select modes, so as to work in concert with highly adaptable and tunable pictures. Although H.264/AVC has been optimized in terms of the deblocking algorithm, it still contributes one-third of the total computational complexity of the decoder.[5]

The hardware reported in Refs. 6–9 applies a traditional processing flow, i.e., firstly, all the pixel signals necessary to process a macroblock (MB) are loaded into the design, and then they are transferred back to the picture memory. Huang *et al.*[6] proposed a typical architecture, wherein horizontal filtering can be processed perfectly, while in the vertical direction, transposed memory must be introduced to avoid the possibility of a memory crash; this process takes nearly twice the computing time in Ref. 7. However, it only considered the calculation of the luminance part of the MB, while the color part was not considered, which is considered to be a limitation of this study. Our newly proposed memory architecture is based on the concept of two-dimensional memory processing introduced by Li *et al.*,[8] wherein the pixel data originally stored in the same memory module is interleaved, so as to avoid the above-mentioned memory crash in Ref. 6. Indeed, a certain hardware cost is required, but such processing is expected to reduce the use of transposed memory. Venkatraman *et al.*[9] suggested

that two sets of deblocking filter modules can be applied at the same time, which can enable the parallel operation of horizontal and vertical filtering, thus reducing the operation time. This however, results in a higher cost in terms of the memory and hardware circuit in the system. The hardware designs proposed in Refs. 10–15 attempted to adopt a process integrating reading, write-back, and processing, while increasing the hardware usage and reducing the waiting time of the circuit. Cheng *et al*.[10] applied shift registers to preserve the temporary data that requires continuous operation, so as to reduce the time required for data access. In the literature study of Chang *et al*.,[11] pixels that must be filtered are first calculated, and the pixels that do not need to be filtered are not transferred into the hardware, resulting in a variable operation time. By using two-dimensional continuous operations, Sheng *et al*.[12] were able to retain more temporary data and reduce the time required for reading and loading. The disadvantage of their approach is the high memory requirement;[13–15] slice memory was introduced to store the pixel data of horizontal pictures, thereby reducing the time spent reading and loading data, which, however, increases the demand for memory. Zheng *et al*.[13] attempted to use more internal buffers to reduce the times required for data reading and loading. Liu *et al*.[14] proposed to unify external memory and internal transpose memory to reduce computation time, and Shih *et al*.[15] proposed the design of a fifth-order pipeline filter with a parallel computation to increase the boundary strength (BS). Effective hardware architectures[6–15] have generally aimed to reduce the circuit size, memory cost, and operation time. However, these architectural designs failed to effectively take into account the memory cost and processing speed. In contrast to the previous designs in Refs. 6–15, in this paper, we propose a more effective hardware architecture that enables the H.264/AVC deblocking filter to achieve a lower memory capacity, reduced access requirements, and a reduced number of operation cycles.

In Sect. 2, we outline the deblocking filter algorithm in accordance with H.264/AVC. In Sect. 3, we thoroughly explain our newly proposed effective hardware architecture design. In Sect. 4, we compare the efficiency of the proposed architecture with that of the previous architectures. We conclude this study in Sect. 5.

2. Deblocking Filter Algorithm for H.264/AVC

A deblocking filter is used to eliminate the blocking effect in pictures by applying its algorithm,[5] thereby generating a smoother picture. In the block-based H.264/AVC standard, the blocking effect originates from the use of conversions with 4 × 4 matrices and block motion compensation. Therefore, a deblocking filter is considered as an effective tool for removing the blocking effect. In theory, a deblocking filter can be separated from the system by postfiltering,[5] which is only used to filter displayed pictures. By introducing a deblocking filter in the loop at the encoding end, higher visual quality is expected to be achieved. This is because the reference picture used for motion compensation has been filtered and reconstructed. Another benefit is that adding a deblocking filter to the system allows the image provider to specify the picture quality at the transmission end.[16–20]

The deblocking filter used in the H.264/AVC standard has high adaptability. The large number of thresholds can be used to adjust the strength of the filter in accordance with the picture and image characteristics. Also, the thresholds can be fine-tuned in accordance with

quantization parameters (QPs), because the generation of the blocking effect is directly related to QPs.[4]

2.1 Filtering order

The deblocking filter applied in H.264/AVC carries out processing with the MB as a unit. There are many 4 × 4 blocks within an MB. The deblocking filter first processes the vertical sides of adjacent blocks in the horizontal direction before processing the horizontal sides of adjacent blocks in the vertical direction. After deblocking filtering in both directions in an MB is completed, the next MB is processed. As shown in Fig. 1, deblocking filtering is carried out on all the MBs in the picture in a raster-scan manner until the MBs of the entire picture are filtered. Deblocking filtering requires a maximum of eight pixel values. Taking into account the BS and QP, filtering may not be performed. If the pixel values of up to six points are to be adjusted, the values of up to three pixels ($p2$, $p1$, $p0$, $q0$, $q1$, $q2$) can be changed on each side of the boundary, as shown in Fig. 2.

2.2 BS

For two adjacent 4 × 4 luma component blocks, a parameter ranging between 0 and 4 called the BS is specified in the standard. For two adjacent blocks, the BS is determined from the selection of the intrablock/interblock prediction mode, the motion vector error, and whether the residual value is encoded. If two adjacent 4 × 4 blocks are subjected to intrablock coding, and the two adjacent sides overlap the boundary of the MB, the strongest filtering mode is used and the BS is set to 4. If two adjacent blocks are subjected to intrablock coding, without overlapping the boundary of the MB, the BS is set to 3. If the above conditions have not yet been satisfied, the determination process is continued. If intrablock coding is used by one of the blocks and one block has added a residual value into the code, a medium-intensity filtering mode is applied and the BS is set to 2. If the motion compensation of the two blocks refers to different pictures, or the difference between the moving coordinates of the two blocks is at least one luma, a

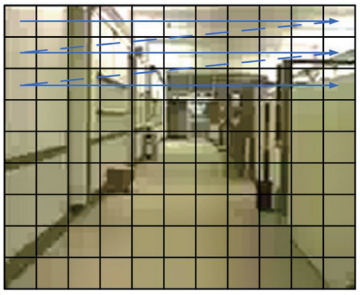


Fig. 1. (Color online) Raster scan.

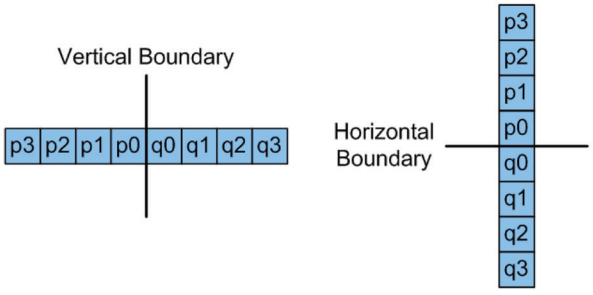


Fig. 2. (Color online) Filtering of vertical and horizontal boundaries.

relatively weak filtering mode is chosen and the BS is set to 1. When all the above-mentioned conditions are not yet satisfied, the boundary remains unfiltered and the BS is set to 0, as shown in Fig. 3. Also, the BS of the chroma component is not to be recalculated; instead, the BS of the corresponding luma component is directly copied to the boundary of the chroma component, as shown in Fig. 4.

2.2.1 Definition of thresholds α and β

The set of sample values to be filtered was mentioned in Sect. 2.2. If the decision of whether or not to carry out filtering is made simply according to the BS, it may result in a blurred picture. Therefore, only the edges with a blocking effect are deblocked, and the other edges remain unchanged so as to preserve the original sharpness of the picture. The decision process is shown in Fig. 5, which indicates that the sample value will be regarded as having a square effect only when the following conditions are met. Thereby, deblocking filtering will be carried out on the sample in accordance with Eqs. (1)–(4).

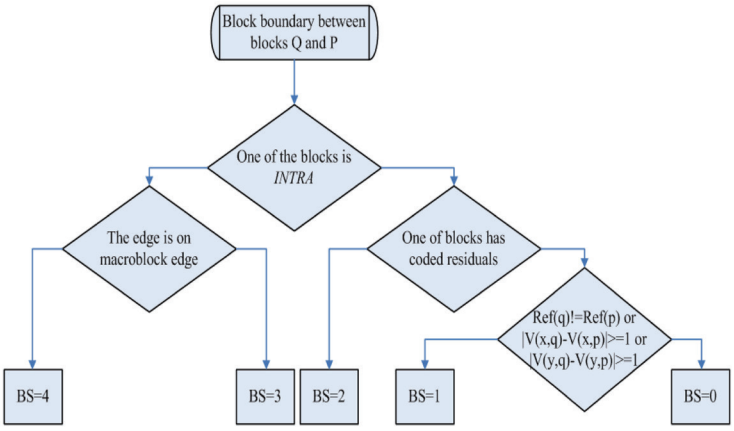


Fig. 3. Decision tree of BS.

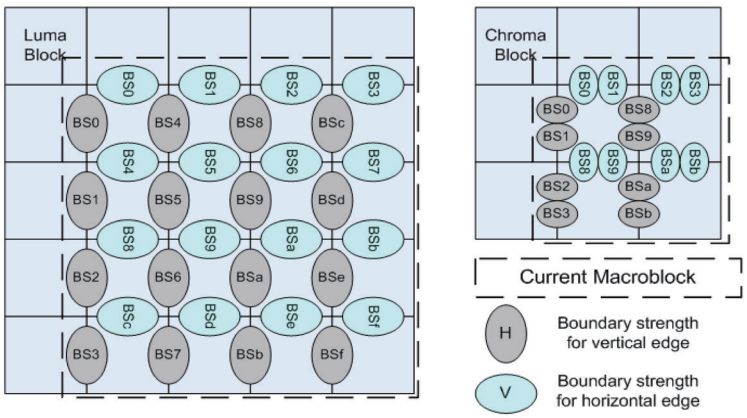


Fig. 4. (Color online) BS of MB at different positions.

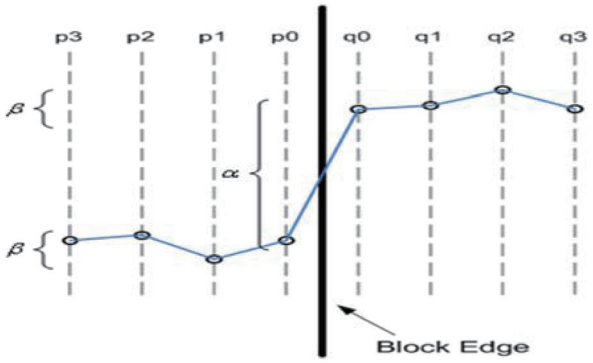


Fig. 5. (Color online) Block edge satisfying the filtering conditions.

$$BS = 1, 2, 3, \text{ or } 4 \tag{1}$$

$$|p0 - q0| < \alpha(Index_A) \tag{2}$$

$$|p0 - p1| < \beta(Index_B) \tag{3}$$

$$|q0 - q1| < \beta(Index_B) \tag{4}$$

With increasing QP of blocks Q and P, the thresholds α and β increase, where α and β are regarded as the criteria for judging whether the changes in the original image are sufficiently large. If the QP shrinks, any changes at the block edge are regarded as the original feature of the picture, not a false edge brought about by the blocking effect. On this basis, α and β are set to small values, to keep the original picture as unchanged as possible. Given that the distortion driven by the block effect becomes rather pronounced when the QP is increased, the inevitable increases in α and β lead to more sampling points in the picture required to apply the deblocking filter.

The slice level of H.264/AVC's is defined by two parameters, which are to adjust offsets at the encoding end, i.e., Offset A and Offset B. These offsets can be fine-tuned at the compression end, thereby causing the filter to apply different thresholds α and β for the same QPs.

$$Index_A = Min(Max(0, QP + Offset_A), 51) \tag{5}$$

$$Index_B = Min(Max(0, QP + Offset_B), 51) \tag{6}$$

2.2.2 Filter mode utilized when *BS* = 1–3

A filter with a basic strength is applied when *BS* = 1–3. The boundary pixels *p*3, *p*2, *p*1, *p*0, *q*0, *q*1, *q*2, and q3 are input to obtain *P*1, *P*0, *Q*0, and *Q*1 via its algorithm. Only the filtered *P*0 and *Q*0 are output, which replace the original *p*0 and *q*0 when they satisfy Eqs. (1)–(4). Only the filtered *P*1 is output, which replaces the original *p*1 when it satisfies Eq. (9). Similarly, *Q*1 is output and replaces the original *q*1 if and only if Eq. (10) is satisfied. The equations used to calculate *P*1, *Q*1, *P*0, and *Q*0 are Eqs. (11) to (14), respectively, where *c*1 is the BS and the coefficient is related to $Index_A$. In the luma component, *c*0 is the number of true values of *c*1 + Eqs. (9) and (10). When considering the chroma component, *c*0 is fixed to *c*1 + 1.

$$|p2 - p0| < \beta(Index_B) \tag{9}$$

$$|q2 - q0| < \beta(Index_B) \tag{10}$$

2.2.3 Filter mode utilized when *BS* = 4

The strongest deblocking filter is applied when *BS* = 4. The pixels *p*3, *p*2, *p*1, *p*0, *q*0, *q*1, *q*2, and *q*3 are input to obtain *P*2, *P*1, *P*0, *P*0*f*, *Q*0*f*, *Q*0, *Q*1, and *Q*2 using the algorithm.

In addition to satisfying Eqs. (1)–(4), if the luma component blocks satisfy Eqs. (9)–(15), the filtered *P*2, *P*1, and *P*0 are output and replace the original pixels. Similarly, *Q*2, *Q*1, and *Q*0 are only output when they are positioned in the luma component and satisfy Eqs. (10) and (15). In the case of the chroma component, if Eq. (9) or (15) is not satisfied by the luma component, then only *P*0*f* out of *p*2–*p*0 replaces *p*0 and is output, with *p*2 and *p*1 remaining unchanged. *Vice versa*, when it comes to the chroma component, or one of Eqs. (10) and (15) in the luma component is not true, then only *Q*0*f* out of *q*0–*q*2 will replace *q*0 and be output, while *q*1 and *q*2 remain unchanged. The formulas for calculating *P*2, *P*1, *P*0, *P*0*f*, *Q*0*f*, *Q*0, *Q*1, and *Q*2 are given as Eqs. (16)–(23).

$$P1 = p1 + CLIP(-c1,\ c1,\ (p2 + ((p0 + q0 + 1) >> 1) - 2p1) >> 1) \tag{11}$$

$$Q1 = q1 + CLIP(-c1,\ c1,\ (q2 + ((p0 + q0 + 1) >> 1) - 2q1) >> 1) \tag{12}$$

$$P0 = p0 + CLIP(-c0,\ c0,\ (4(q0 - p0) + (p1 - q1) + 4) >> 3) \tag{13}$$

$$Q0 = q0 - CLIP(-c0,\ c0,\ (4(q0 - p0) + (p1 - q1) + 4) >> 3) \tag{14}$$

$$|p0 - q0| < ((\alpha(Index_A)) >> 2) + 2 \tag{15}$$

$$P2 = (2p3 + 3p2 + p1 + p0 + q0 + 4) >> 3 \tag{16}$$

$$P1 = (p2 + p1 + p0 + q0 + 2) >> 2 \tag{17}$$

$$P0 = (p2 + 2p1 + 2p0 + 2q0 + q1 + 4) >> 3 \tag{18}$$

$$Q0 = (q2 + 2q1 + 2q0 + 2p0 + p1 + 4) >> 3 \tag{19}$$

$$Q1 = (q2 + q1 + q0 + p0 + 2) >> 2 \tag{20}$$

$$Q2 = (2q3 + 3q2 + q1 + q0 + p0 + 4) >> 3 \tag{21}$$

$$P0f = (2p1 + p0 + q1 + 2) >> 2 \tag{22}$$

$$Q0f = (2q1 + q0 + p1 + 2) >> 2 \tag{23}$$

In these equations, CLIP stands for the reduction operation and >> represents the shift operation.

3. Hardware Architecture Design

In Fig. 6, an architecture for the deblocking filter system is proposed. In the memory, interleaved methods are applied to store pixels, thereby addressing the problem of transposing memory in previous designs. In addition, two parts of modularized memory with different functions are proposed, equipping both memory modules with the capability of two-dimensional storage. One of the memory modules is centered on the core of the dual-port static random access memory (SRAM), storing the block data under real-time processing, which is referred to as the RAM-0 module. The other memory module is in the mode of the two-port SRAM, which stores the block data on the far right of the previous MB and is named the RAM-1 module. A novel feature of this architecture is that a fourth-order pipeline filter is placed in

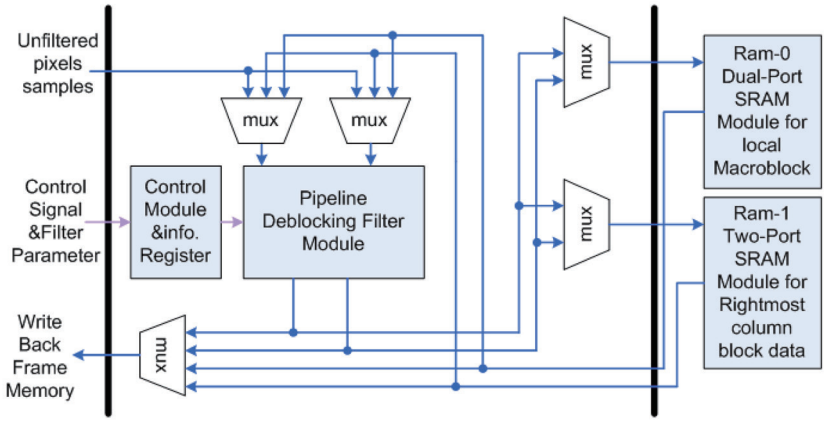


Fig. 6. (Color online) Architecture for the proposed deblocking filter system.

the deblocking filter module, which, coupled with the proposed recursive control, can reduce the frequency of memory access. In the control unit, in addition to controlling the addresses of the basic memory components, the data stream selection, and the data input and output, the parameters required for the deblocking filter module must also be stored in a system on chip (SOC). These parameters include $Index_A$, $Index_B$, and the BS. In the system architecture diagram shown in Fig. 6, except for the control signal lines, the width of the entire internal data bus is 32 bits. The external bus input and output pixel data are asynchronous. Therefore, they can share one 32-bit bidirectional channel, making this architecture ideal for an SOC.

Four 4 × 4 luma component blocks and four 4 × 4 color component blocks from an MB are stored in the module RAM-1, while the data is composed of eight groups of 16 word × 8 bit dual-port memory, for which the required storage space is 32 × 32 bits, equivalent to 1024 bits. The other memory module RAM-0 temporarily stores the pixel data of the MB while it is being processed. For the deblocking filter processing flow proposed in this paper, the maximum temporary storage is sixteen 4 × 4 blocks; the RAM-0 module consists of eight groups of 32 word × 8 bit dual-port memory, for which the storage space required is 64 × 32 bits, equivalent to 2048 bits.

For the deblocking filter architecture proposed in this paper, the processing of an MB only takes 279 computation cycles, among which five cycles are required to load the calculated BS and the parameters $Index_A$ and $Index_B$, and the other 274 cycles are required for loading pixels and restoring data. When the processing of an MB has reached the last MB in the picture, an additional 32 cycles are required to store the pixel data remaining in RAM-1 in the memory of the reconstructed picture, giving a total of 301 computation cycles.

3.1 Data processing flow

The deblocking filter signal processing proposed in this paper is shown in Figs. 7 and 8, in which B0–B39 are 4 × 4 pixel blocks. The circles numbered 1 to 48 all require four

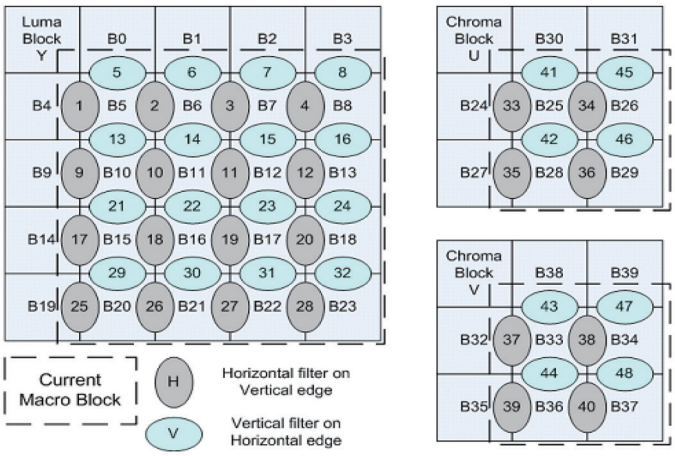


Fig. 7. (Color online) Data processing sequence.

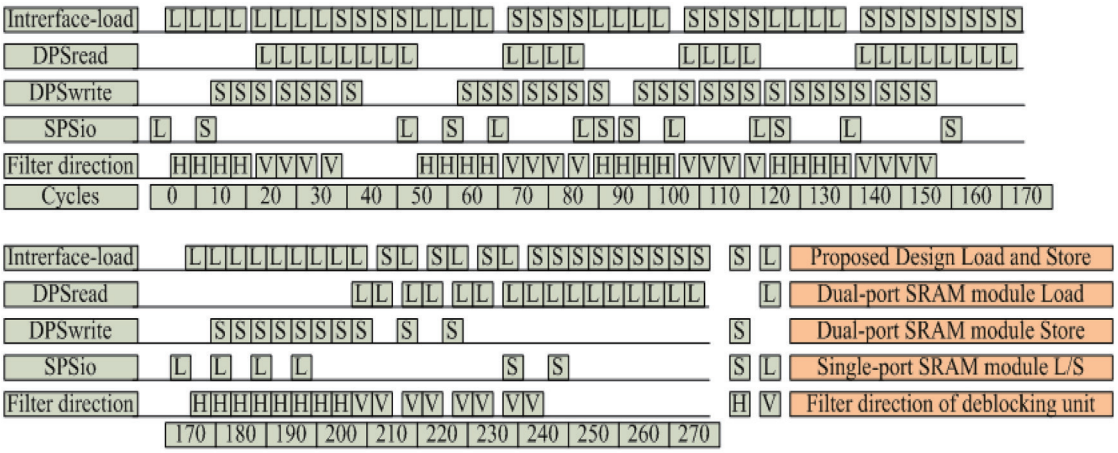


Fig. 8. (Color online) Time sequence of deblocking filter.

computation cycles; ellipse H indicates horizontal filtering of the vertical edges and ellipse V indicates vertical filtering of the horizontal edges. In the first stage, the data in block B5 is received from the outside, and the data in block B4 is extracted from the RAM-1 module and sent to the deblocking filter module. In the second stage, the block filter will successively complete the B4 and B5 obtained from the first stage in a synchronized manner. At this point, because B5 must be input in the deblocking filter again, it is not necessary to write B5 in the memory. Instead, it is necessary to send B5 and B6 received from the outside to the deblocking filter module. At the same time, B4 is temporarily saved in the RAM-0 module. In the fifth stage, the data in block B0 data is received from the outside, and the data in block B5 stored in the RAM-0 module in the second stage is extracted and sent to the deblocking filter module. Moreover, B7 and B8 are stored together in the RAM-0 module. After stage eight has been completed, B0–B3, which are not used again in the current MB processing flow, are extracted and sent to the outside of the deblocking filter module. After completing stages 16, 24, 32, 45, 46, 47, and 48, blocks B8, B13, B18, B23, B26, B29, B34, and B37 are respectively stored in the RAM-1 module, because the next MB is soon applied again, thereby the time spent in the read and load stages is reduced, and the repeated loading and writing of the same block can be eliminated.

3.2 Internal memory planning

In Fig. 9, during the vertical access of the memory, memory crash occurs, which hinders the access from being completed in one execution cycle. This means that transposed memory must be used, which can result in reduced efficiency. To solve this problem, we employ the two-dimensional memory access design proposed in Ref. 8, which applies interleaved methods for data placement in different memory modules, eliminating the origin of the memory crash in the previous design. It also makes it possible to read and write in both the horizontal and vertical directions.

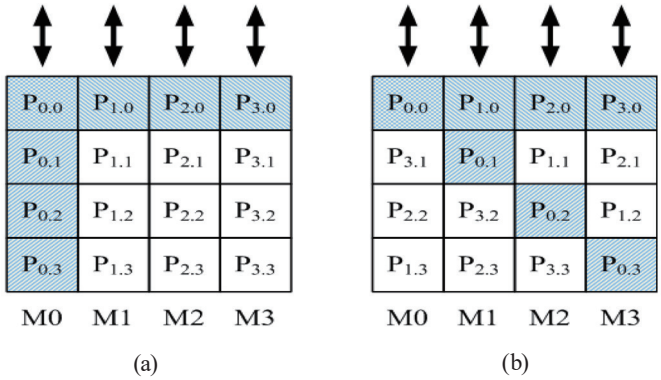


Fig. 9. (Color online) Diagram of memory access. (a) Traditional memory storage and (b) shift transposition storage method.

In a two-dimensional memory module, considering that vertical pixel data is placed in different modules, memory crash can be entirely avoided, allowing both horizontal and vertical operations to be accelerated in two-dimensional memory without the occurrence of memory crash.

To achieve two-dimensional memory access, compatible components are required for the circuit to conduct data segmentation, address generation, and data combination, as shown in Fig. 10. The address generator produces the address corresponding to the read or write of the memory row or column as required. Also, the module in which data segmentations occurs is responsible for shifting the input pixel and determining the frequency of the data shift for different input addresses, so as to enable vertical pixel signals to be stored in different memory modules. The data combination module receives the shifted data and the output from the memory unit, and according to the output address, the data combination module reverses the shift by restoring the pixel data in the memory module back into the pixel data that has not been previously shifted. Thereby, the correct sequence can be obtained, facilitating the subsequent processing.

3.3 Pipeline-based deblocking filter module

To optimize the processing performance of the deblocking filter, a parallel and pipelined circuit design is used as shown in Fig. 11. The parallel processing data includes eight pixel inputs, eight pixel outputs, and a set of recursive inputs, and the reduced latency of the critical path is largely avoided by employing a fourth-order pipeline design. Different signals are selected as the input source to read pixels. The purpose of Stage 1 is to query the values of α, β, and Clip in line with the values of $Index_A$, $Index_B$, and BS, respectively, so as to conduct initial processing on the input pixels. In Stage 2, the output from Filter Stage 1 is processed and the precalculation of the decision flag required by the final signal selector is conducted. Stage 3 performs the calculation in the last stage using the output result obtained from the part of Filter

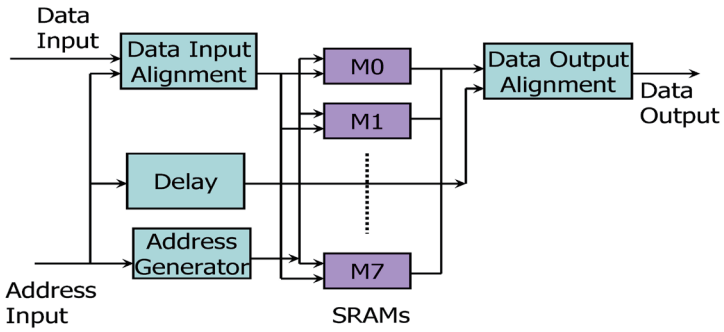


Fig. 10. (Color online) Schematic diagram of memory modules.

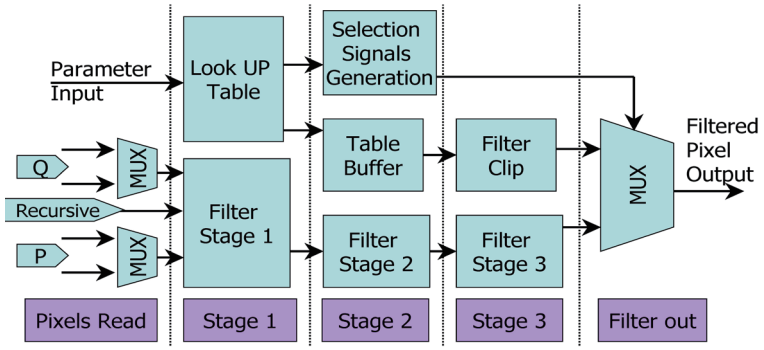


Fig. 11. (Color online) Architecture diagram of pipeline deblocking filter module.

Stage 2 with $BS = 4$. Filter Clip applies the reduction operation to the output result obtained from the part of the Filter Clip with the BS ranging from 1 to 3. In the Filter Out stage, the final result based on the previously calculated decision flag is selected and output.

3.3.1 Flag operation unit

On the basis of the absolute value obtained by subtracting the real-time BS from the pixel values, a flag operation unit can determine whether the boundary on the picture is a false boundary generated by the block effect or a real one sourced from an actual picture, from which *FLAG*1 to *FLAG*6 and *FLAG* chroma can be obtained. In the case of different BSs, the flag is used for decision making [Eqs. (24)–(26) are all satisfied]. If a false boundary resulting from the blocking effect is spotted, the filtered pixels are output to remove the effect. If the result is the boundary of a real picture's boundary [one of Eqs. from (24)–(26) is not satisfied], or the BS is 0, the original pixel can be left unfiltered. Output selection lists are detailed in Tables 1 and 2.

$$FLAG1 = | p0 - q0 | < \alpha(Index_A) \tag{24}$$

$$FLAG2 = | p1 - p0 | < \beta(Index_B) \tag{25}$$

Table 1
Output selection table when BS = 1–3.

Filtered pixels	Flag1	Flag2	Flag3	Flag4	Flag5	Flag chroma	Output symbol
$P1$	True	True	True	True	X	False	$bs1p1$
$P0$	True	True	True	False	False	False	$bs1p0$
$P0$	True	True	True	True	False	False	$bs1p0a1$
$P0$	True	True	True	False	True	False	$bs1p0a1$
$P0$	True	True	True	True	True	False	$bs1p0a2$
$P0$	True	True	True	X	X	True	$bs1p0a1$

Table 2
Output selection table when BS = 4.

Filtered pixels	Flag1	Flag2	Flag3	Flag4	Flag5	Flag6	Flag chroma	Output symbol
$P2$	True	True	True	True	X	True	False	$bs4p2$
$P1$	True	True	True	True	X	True	False	$bs4p1$
$P0$	True	True	True	True	X	True	False	$bs4p0$
$P0$	True	True	True	False	X	True	False	$bs4p0f$
$P0$	True	True	True	True	X	False	False	$bs4p0f$
$P0$	True	True	True	X	X	X	True	$bs4p0f$

$$FLAG3 = |q1 - q0| < \beta(Index_B) \tag{26}$$

$$FLAG4 = |p2 - p0| < \beta(Index_B) \tag{27}$$

$$FLAG5 = |q2 - q0| < \beta(Index_B) \tag{28}$$

$$FLAG6 = |p0 - q0| < (((\alpha(Index_A)) >> 2) + 2) \tag{29}$$

$$FLAG\ Chroma = (y = 0)\ \text{or}\ (u\ \text{or}\ v = 1) \tag{30}$$

3.3.2 When BS = 1–3

"True", "False", and "X" in the following tables represent true, false, and unaffected, respectively. Taking $P1$ as an example, when $FLAG1$, $FLAG2$, $FLAG3$, and $FLAG4$ are all true and $FLAG\ Chroma$ is false, then the output of $P1$ is the value ($bs1p1$) processed by the filter; otherwise, it remains unchanged.

$$bs1p1 = p1 + clip(-c1, c1, ((p2 + ((p0 + q0 + 1) >> 1) - 2p1) >> 1)) \tag{31}$$

$$bs1p0 = p0 + clip(-c0, c0, (((q0 - p0) << 2) + (p1 - q1) + 4) >> 3)) \tag{32}$$

3.3.3 When *BS* = 4

$$bs4p2 = ((2p3 + 3p2 + p1 + p0 + q0 + 4) >> 3) \tag{33}$$

$$bs4p1 = ((p2 + p1 + p0 + q0 + 2) >> 2) \tag{34}$$

$$bs4p0 = ((p2 + 2p1 + 2p0 + 2q0 + q1 + 4) >> 3) \tag{35}$$

$$bs4p0f = ((2p1 + p0 + q1 + 2) >> 2) \tag{36}$$

3.3.4 Operation decomposition of operation for pipeline filter

According to the deblocking filter algorithm proposed in H.264/AVC, the operation equation contains many common items. Therefore, we can further optimize the decomposition while carrying out hardware design and implementation, which can also facilitate the segmentation of the pipeline filter. For example, Eqs. (33)–(36), for which *BS* = 4, can be replaced by the following equations:

$$bs4p2 = (Sp0q0p1p2 + (Sp2p3 << 1)) >> 3; \tag{37}$$

$$bs4p1 = (Sp0q0p1p2) >> 2; \tag{38}$$

$$bs4p0 = (Sp0q0p1q1 + Sp0q0p1p2) >> 3; \tag{39}$$

$$bs4p0f = (Sp0p1 + Sp1q1) >> 2; \tag{40}$$

Taking Eqs. (10) and (11) with the BSs of 1, 2, and 3 as an example, we derived the following procedures:

$$bs1p1 = (Sp0q0 + \{Sd2p1p2[8], Sd2p1p2, 1'b0\}) >> 2; \tag{41}$$

$$bs1p0 = ((Sq0dp0 << 2) + \{\{Sp1dq1[9]\}, Sp1dq1\} + 11'd4) >> 3; \tag{42}$$

$$bs1q1 = (Sp0q0 + \{Sd2q1q2[8],\ Sd2q1q2,\ 1'b0\}) >> 2; \tag{43}$$

A number of common items are involved in Eqs. (37)–(43), which can be extracted and further simplified as shown below:

$$Sp0q0 = p0 + q0 + 1; \tag{44}$$

$$Sp0p1 = p0 + p1 + 1; \tag{45}$$

$$Sp1p2 = p1 + p2 + 1; \tag{46}$$

$$Sp2p3 = p2 + p3 + 1; \tag{47}$$

$$Sp0q0p1q1 = Sp0q0 + Sp1q1; \tag{48}$$

$$Sp0q0p1p2 = Sp0q0 + Sp1p2; \tag{49}$$

$$Sp0q0q1q2 = Sp0q0 + Sq1q2; \tag{50}$$

$$Sq0dp0 = q0 - p0; \tag{51}$$

$$Sp1dq1 = p1 - q1; \tag{52}$$

$$Sd2p1p2 = (p2 - (p1 << 1)); \tag{53}$$

$$Sd2q1q2 = (q2 - (q1 << 1)); \tag{54}$$

where Eqs. (44)–(47) and (51)–(54) can be completed in the Filter 1 stage, Eqs. (40)–(43) and (48)–(50) can be completed in Filter Stage 2, and Eqs. (37) and (39) can be carried through to Filter Stage 3. The trimming functions required for Eqs. (8)–(11) can be completed during the Filter Clip.

4. Simulation Results and Comparison

The hardware circuit design for the proposed architecture was based on VerilogHDL and synthesized using Synopsys Design Compiler under a TSMC CMOS 0.18 μm process with the operating frequency set to 100 MHz. The number of synthesized logic gates was 19.4 K. By comparing the proposed hardware architecture with those in the recent literature,[6–15] we avoided the disadvantages of the deblocking filter in terms of the memory cost and processing speed, as shown in Table 3.

We hereby briefly expounded on the fields that were evaluated. Cycles/MB is defined as the number of execution cycles required to fully filter an MB. Filter cycle/MB is the number of operation cycles actually needed by the deblocking filter in the core. SRAM for pixels is the memory applied in the design, and SRAM in bits is the memory in bits used to directly convert the design via the capacity that can be stored, where the space involved in this part is not included in the gate count. 4 × 4 registers consist of the internal memory of the circuit, first-in, first-out (FIFO), the transposed memory, and space. Registers in bits are the storage space that is converted to the corresponding bits; # of edge filters is the number of deblocking filter cores utilized. The calculation of frames per second (FPS) can produce the number of frames of 1280 × 720 size that can be processed per second at the same frequency of 100 MHz;

Table 3
Comparison of hardware circuit designs.

Method	Ref. 10	Ref. 11	Ref. 12	Ref. 13	Ref. 14	Ref. 15
Cycles (MB)	614	584	566	510	386	Max 374 Min 50 Avg. 86–244
Filter cycle (MB)	240	214	192	136	336	0–374
SRAM for pixels	Dual 96 × 32 Dual 64 × 32	Dual 16 × 32	8 Dual 80 × 8	Dual 88 × 32 Dual 72 × 32 Single 32 × 32	Single 80 × 32	Single 96 × 32
SRAM in bits	5120	512	5120	6144	2560	3072
#4 × 4 registers	4	6	0	11	2	2
Registers in bits	512	768	0	1408	256	256
# of edge filters	1	1	1	2	1	1
Capability @1280*720p@ 100 MHz	45 FPS	47.5 FPS	49.1 FPS	54.46 FPS	71.9 FPS	113.8–322.9 FPS
Process (μm)	.25	N/A(FPGA)	.35	N/A(FPGA)	.18	.18
Gate count	20.66k	N/A(FPGA)	9.35k	N/A(FPGA)	21.8k	20.9k
Performance	3458048	750080	2897920	3851520	1086976	1244672

Method	Ref. 16	Ref. 17	Ref. 18	Ref. 19	Proposed
Cycles (MB)	446	238	250	214	279
Filter cycle (MB)	192	236	250	200	277
SRAM for pixels	Dual 64 × 32 2 Two 96 × 32	Dual 96 × 32 Dual 64 × 32 Single (2 × frame width) × 32	2 Single 96 × 32 Single (2 × frame width +20) × 32	Single 96 × 32 Two 32 × 32 Single (1.5 × frame width) × 32	8 Dual 32 × 8 8 Two 16 × 8
SRAM in bits	8192	5120 + 65536	6144 + 66176	4096 + 49152	3072
#4 × 4 registers	8	9	4	2	0
Registers in bits	1024	1152	512	256	0
# of edge filters	1	1	1	1	1
Capability @1280*720p@ 100 MHz	62.3 FPS	97.1 FPS	111.1 FPS	128.6 FPS	98.15 FPS
Process (μm)	.25	.18	.18	.18	.18
Gate count	24k	19.5k	19.6k	20.9k	19.4k
Performance	4110336	1806336 20680704	1664000 18208000	931328 11449856	857088

taking into account the design process, the cost of the circuit after synthesis, and the figure of merit derived by the performance measurement system P (performance = cycle × memory), the obtained reference factors are in line with the standard. We carried out a comparison of the proposed hardware architecture with those in the recent literature, as summarized in Table 3.

5. Conclusion

In this paper, a new hardware architecture design was proposed, which employs a deblocking filter with low cost and high efficiency. The distinguishing feature of the proposed hardware

architecture is that it is based on a static random-access memory module, its application of a fourth-order pipeline deblocking filter architecture, and its improved read/load capability, reducing the computation time needed and the frequency of memory access. In terms of integrated circuits, the architecture design was carried out with Verilog HDL and the circuit synthesis tool Synopsys Design Compiler under a TSMC CMOS 0.18 μm process.

In contrast to the architectures proposed in the recent literature, the deblocking filter proposed in this study applies two-dimensional access memory for the interleaving of data, which enables the design to be free from transposed memory. This approach is characterized by a low memory requirement (3072 bits) and as few as 279 operation cycles to calculate an MB. The experimental results suggest that our newly designed hardware architecture is compatible with the H.264/AVC video compression system and can satisfy the requirements for real-time computation.

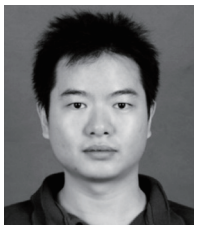
Acknowledgments

This work is supported by the young teachers' education scientific research project of Fujian province (Grant no. JT180455), Xiamen Science and Technology Foundation (Grant 3502Z20173035), and Fujian Provincial Natural Science Foundation of China (Grant 2018J01570).

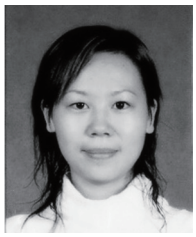
References

1 T. Wiegand: ITU-T Rec. H. 264| ISO/IEC 14496-10 AVC, JVT-G050 (2003).
2 T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra: IEEE Trans. Circuits Syst. Video Technol. **13** (2003) 560.
3 JVT H.264/AVC Reference Software JM 9.5.
4 I. E. Richardson: H. 264 and MPEG-4 Video Compression (Wiley, 2004).
5 P. List, A. Joch, J. Lainema, G. Bjontegaard, and M. Karczewicz: IEEE Trans. Circuits Syst. Video Technol. **13** (2003) 614.
6 Y. Huang, T. Chen, B. Hsieh, T. Wang, T. Chang, and L. Chen: Proc. Int. Conf. Multimedia and Expo 1 (2003) 1–693.
7 M. Sima, Y. Zhou, and W. Zhang: IEEE Trans. Consum. Electron. **50** (2004) 292.
8 L. Li, S. Goto, and T. Ikenaga: IEICE Trans. Inf. Syst. **88** (2005) 1623.
9 V. Venkatraman, S. Krishnan, and N. Ling: Picture Coding Symp. (2004) 1623.
10 C. Cheng and T. Chang: IEEE Int. Conf. Consumer Electronics, Digest of Technical Papers (2005) 235.
11 S. Chang, W. Peng, S. Wang, and T. Chiang: IEEE Trans. Consum. Electron. **51.1** (2005) 249.
12 B. Sheng, W. Gao, and D. Wu: IEEE Int. Conf. Image Proc. **1** (2004) 665.
13 V. Venkatraman, S. Krishnan, and N. Ling: Proc. Picture Coding Symp., San Francisco, California, USA, Dec. (2004).
14 T. Liu, W. Lee, T. Lin, and C. Lee: IEEE Int. Symp. Circuits and Systems (2005) 2140.
15 S.-C. Chang, W.-H. Peng, S.-H. Wang, and T. Chiang: IEEE Trans. Consum. Electron. **51** (2005) 249.
16 MPEG Video Group: MPEG-4 Video Verification Model Version 8.0, ISO/IEC JTC1/SC29/WG11 N1796, July 1997.
17 A. Luthra, G. J. Sullivan, and T. Wiegand: IEEE Trans. Circuits Syst. Video Technol. **13** (2003) 557.
18 J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi: IEEE Circuits Syst. Mag. **4** (2004) 7.
19 K. Denolf, C. Blanch, G. Lafruit, and A. Bormans: IEEE Workshop Signal Proc. Systems (IEEE, 2002) 222.
20 M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro: IEEE Trans. Circuits Syst. Video Technol. **13** (2003) 704.

About the Authors



Xiaodong Zheng is a lecturer and teacher in the School of Software Engineering at Xiamen University of Technology. His research interests lie in the areas of software development, information security, and artificial intelligence. (zxd@xmut.edu.cn)



Wei Zhu is a college lecturer in the School of Software Engineering at Xiamen University of Technology. Her research interests lie in the areas of software development, information security, and artificial intelligence. (zhw@xmut.edu.cn)

Shaoyong Yu received his Ph.D. degree from Xiamen University in 2017. He is also a college lecturer in the School of Mathematics and Information Engineering, Longyan University. His research interests lie in the areas of computer vision and deep learning. (syyu@xmut.edu.cn)

Jinpeng Wu is a lecturer and teacher in the School of Software Engineering at Xiamen University of Technology. His research interests lie in the areas of probability theory and mathematical statistics. (wjp@xmut.edu.cn)