Journal of Zhejiang University SCIENCE A
ISSN 1009-3095 (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Macroblock-level decoding and deblocking method and its pipeline implementation in H.264 decoder SOC design*

WANG Shu-hui†, LIN Tao, LIN Zheng-hui

(*Department of Electronic Engineering, Shanghai Jiao Tong University, Shanghai 200030, China*)

†E-mail: wangshuhui_cn@yahoo.com.cn

Received Apr. 4, 2006; revision accepted July 6, 2006

Abstract: This paper presents a macroblock-level (MB-level) decoding and deblocking method for supporting the flexible macroblock ordering (FMO) and arbitrary slice ordering (ASO) bit streams in H.264 decoder and its SOC/ASIC implementation. By searching the slice containing the current macroblock in the bit stream and switching slices correctly, MBs can be decoded in the raster scan order, while the decoding process can immediately begin as long as the slice containing the current MB is available. This architectural modification enables the MB-level decoding and deblocking 3-stage pipeline, and saves about 20% of SDRAM bandwidth. Implementation results showed that the design achieves real-time decoding of 1080HD (1920×1088@30 fps) at a system clock of 166 MHz.

Key words: Flexible macroblock ordering (FMO), Arbitrary slice ordering (ASO), System-on-chip (SOC), Raster scan order, Pipeline

doi:10.1631/jzus.2007.A0036 **Document code:** A **CLC number:** TN919.81

INTRODUCTION

H.264 (Wiegand *et al.*, 2003) is the latest video coding standard developed by the ISO/IEC Moving Picture Experts Group (MPEG) and the ITU-T Video Coding Experts Group (VCEG). Besides many new technologies which can bring high compression efficiency, H.264 also provides other tools to flexibly adapt to different transmission networks, such as flexible macroblock ordering (FMO) (Wenger and Horowitz, 2002a; 2002b) for dealing with data errors/losses in error prone environments, arbitrary slice ordering (ASO) for networks having a feature known as network jitter, etc. FMO and ASO are allowed in Baseline and Extended profiles. Although FMO enables more efficient error concealment and ASO improves end-to-end delay in real-time applications, both of them on the other hand increase decoder complexity.

Current researches for the implementation of H.264 decoder can generally be divided into two categories: Khan *et al.*(2004), Iverson *et al.*(2004), Lee *et al.*(2004) and Ramadurai *et al.*(2005) focused on software-based architecture, while Ha *et al.*(2004), Kang *et al.*(2004), Chen *et al.*(2005), Park *et al.*(2005) and Lee *et al.*(2006) are application-specific integrated circuit (ASIC)-based solution. Our decoder is an ASIC-based solution and supports Baseline profile and Main profile up to level 4 (1920×1088@30 fps). For the FMO and ASO bit streams, Ha *et al.*(2004), Park *et al.*(2005) and Lee *et al.*(2006) did not consider this case, Kang *et al.*(2004) adopted the same two-pass decoding and deblocking method as H.264 reference decoder, while Chen *et al.*(2005) switched the deblocking pipeline schedule from macroblock-level (MB-level) to frame-level, which increases the implementation complexity and hurts the decoder performance. Differing from the previous literature, in this paper, we propose an approach to decode MB in the raster scan order when FMO and ASO are adopted in H.264 decoder, and the decoding and de-

* Project (No. 2002AA1Z1190) supported by the National Hi-Tech Research and Development Program (863) of China

blocking can be performed in pipeline on an MB-by-MB basis.

IMPACT OF FMO AND ASO ON H.264 DECODER

In H.264, when FMO is not used, a picture is divided into one or several slices, and the MBs are placed in slices in the raster scan order. FMO extends the concept of slices by allowing non-consecutive MBs to be placed in the same slice. When using FMO, the MBs in a picture are divided into up to 8 slice groups, each of which is composed of one or several slices. However, the MB addresses within a slice are still in the ascending order. MB allocation map (MBAmap) has 7 different types, such as interleaved slices, dispersed MB allocation, etc. The MBAmap consists of one integer per MB of the picture, which indicates the slice group id (SGid). When ASO is enabled, the slices in the bit stream may arrive at the decoder side in an arbitrary order. From an implementation complexity point of view, ASO is roughly comparable with FMO.

In H.264, deblocking is performed on an MB basis, with all MBs in a picture processed in the raster scan order. Therefore, MBs in the FMO/ASO bit stream is not in the same order as deblocking should be performed. It is very challengeable for the decoder implementation. When FMO and ASO are used, H.264 reference decoder decodes the slices in the receiving order, which results in the following problems: (1) The deblocking of MB cannot be carried out until the entire picture is first decoded and stored completely. Thus, this lack of concurrency means that decoding and deblocking tasks have to run faster. (2) The newly reconstructed picture has to be stored first, and later swapped in for deblocking. As a result, the necessary SDRAM bandwidth is roughly doubled for data transfer of current picture.

PROPOSED SCHEME OF MB-LEVEL DECODING AND DEBLOCKING

In order to do MB decoding and deblocking in pipeline on an MB-by-MB basis, it is necessary to ensure MB decoding in the raster scan order. When FMO and ASO are used, we can do it as follows: (1) Set up an incremental counter of MB address; (2) Locate the slice including the current MB; (3) If the previous MB and the current MB are in different slices, switch to the slice containing the current MB.

In H.264 reference decoder, before decoding the current picture, we can get the SGid for every MB from the *MbToSliceGroupMap*[] array. Meanwhile, the first MB address of each slice can be obtained from the *first_mb_in_slice* syntax element in the slice header. By using *MbToSliceGroupMap*[] array and *first_mb_in_slice* of each slice, the slice containing the current MB can be found correctly.

Fig.1 shows the flowchart of the MB-level decoding and deblocking. Normally, there are two approaches to implement the MB-level decoding and deblocking. In the first approach, decoding process can start only after reading all the slices of the current picture and in the second one, once the slice including

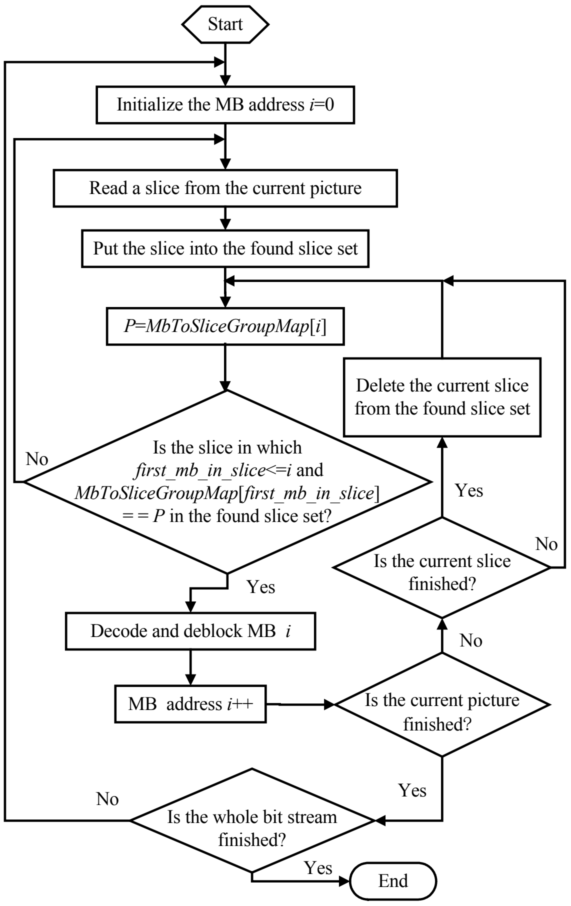
Start
Initialize the MB address i=0
Read a slice from the current picture
Put the slice into the found slice set
P=MbToSliceGroupMap[i]
Delete the current slice from the found slice set
No
Is the slice in which first_mb_in_slice<=i and MbToSliceGroupMap[first_mb_in_slice] = = P in the found slice set?
Yes
No
Is the current slice finished?
Yes
Decode and deblock MB i
No
MB address i++
Is the current picture finished?
No
Yes
Is the whole bit stream finished?
Yes
End

Fig.1 Flowchart of the MB-level decoding and deblocking

the current MB is found, the decoding process can immediately begin. Understandably, depending on the design of MBAmap, the second approach has less delay between decoding and display than the first one. In this decoder, the latter is used, and in order to reduce SDRAM cost and D-cache access penalty, two slice header buffers with different sizes are allocated to keep the slice header information. One is used for storing the common information for different slices, while the other is used for storing other information which varies with slices, such as the first MB address, quantization parameter *QP*, etc. Similarly, the slice data content is put in the slice data buffer. To switch among these slices exactly, a list of pointers is used to indicate the slice header position in the second buffer and the slice data position in the slice data buffer.

Fig.2 is the block diagram of the MB-level decoding and deblocking. We can divide the project into several modules.

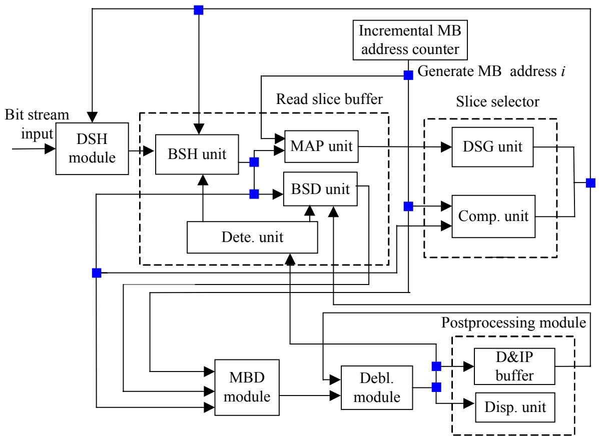

Fig.2 Block diagram of MB-level decoding and deblocking

(1) Module of decoding slice header (DSH module) parses the slice header information in the bit stream.

(2) The read slice buffer is used to keep the contents of the slices that have been read out from the bit stream, and includes Unit for buffering slice header (BSH unit), Unit for buffering slice data (BSD unit), MAP unit and Unit for determining whether one slice is finished (Dete. unit). Hereinto, MAP unit keeps the MBAmap information and computes the SGid of each MB in one picture.

(3) Incremental MB address counter generates the incremental MB address *i*.

(4) Slice selector locates the slice containing the current MB *i*, and consists of Unit for determining slice group (DSG unit) and Unit for comparing the *first_mb_in_slice* in one slice with *i* (Comp. unit). The two units are used for searching the slice containing the current MB in the read slice buffer. If such slice already exists, then the decoder decodes the current MB; otherwise, the decoder continues to read the slices in the bit stream till such slice is found.

(5) MB decoding (MBD) module reconstructs the current MB after slice selector finds the slice containing the current MB.

(6) Deblocking (Debl.) module filters the reconstructed MB.

(7) Postprocessing module includes Buffer used for deblocking and intra prediction (D&IP buffer) and Unit used for displaying or storing picture (Disp. unit). Here, D&IP buffer keeps the data to be used in the deblocking and intra prediction of the succeeding MBs.

ASIC IMPLEMENTATION OF MB-LEVEL DECODING AND DEBLOCKING

MB-level decoding and deblocking pipeline

The complete decoding system is shown in Fig.3. The dual parallel bus (CPU System Bus and High-Speed Memory Bus) architecture is used to satisfy the requirement of high data bandwidth. Prediction (include intra and inter prediction) and deblocking&storing modules are implemented in dedicated hardware, and other modules, such as CAVLC (context-based adaptive variable length coding)/ CABAC (context-based adaptive binary arithmetic coding) and IQ&IT (inverse quantization and inverse transform), are implemented by a 32 bit RISC CPU.

Software part works concurrently with hardware part. In the design, the main function of storing module includes: (1) Storing the deblocked picture data. (2) Access to temporary picture data for deblocking of the succeeding MBs. By using the MB-level decoding and deblocking method proposed in Section 3, deblocking module directly receives the MB data from the prediction module. Considering the unfixed time consumed by prediction module (especially for inter prediction) and the similarly frequent

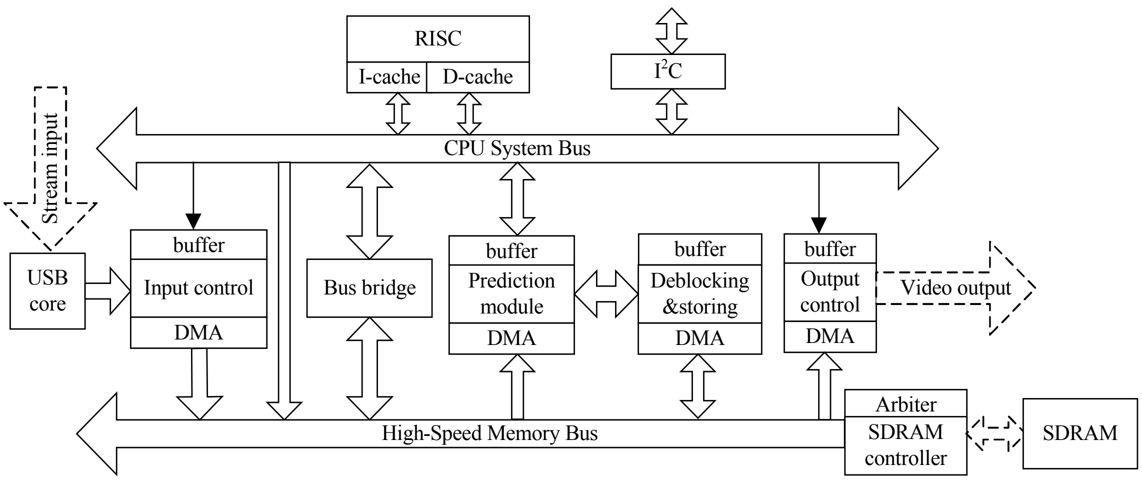

Fig.3 Architecture diagram of H.264 decoder

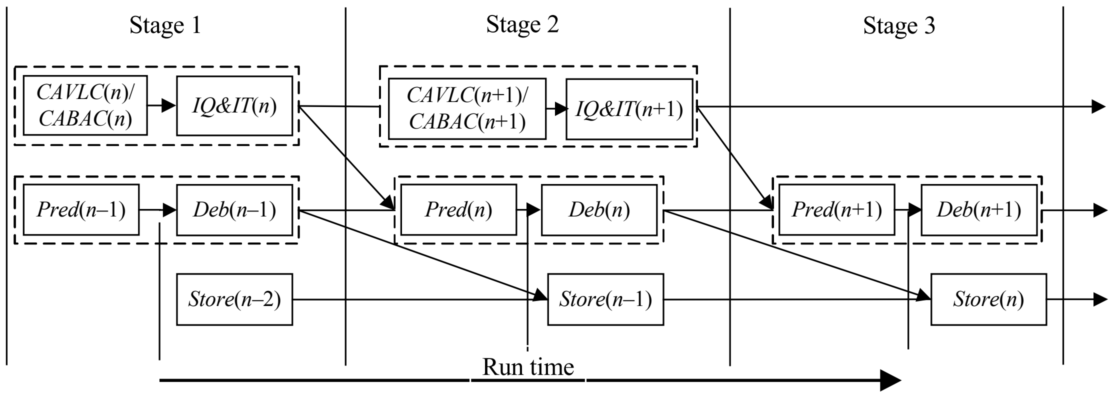

Fig.4 Pipeline stages of MB-level decoding and deblocking

SDRAM access of prediction module and deblocking&storing module, prediction module runs serially with deblocking&storing module, while deblocking and storing operations are parallel, and the two operations take similar cycles in this decoder, thus arranging specialized storing module can greatly improve decoder performance. The 3-stage pipeline schedule for the decoder is shown in Fig.4, *Pred*(n) and *Deb*(n) represent prediction and deblocking operations of the nth MB, respectively.

Storage of temporary data for deblocking and intra prediction

In H.264, all MBs in a picture are filtered in the raster scan order, and deblocking is applied to all 4×4 block edges of a picture, except the edges at the boundary of the picture. When filtering each MB, the deblocked samples of the above and left MBs of the current MB must be available, as shown in Fig.5.

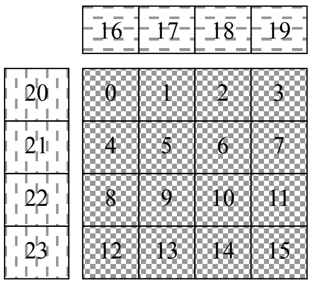

Fig.5 Data used in deblocking one MB

In Fig.5, the above 4×4 blocks (16~19) and the left 4×4 blocks (20~23) are needed in the deblocking process of the current MB. Therefore, we need to store the left four 4×4 blocks and the above one line of 4×4 blocks of the current MB in the decoding process. For YCbCr 4:2:0 video, the two chroma components (Cb and Cr) each has half the horizontal and vertical resolution of luma component (Y). Thus, the buffer used to store the left 4 blocks (LB) is (16×4×8)+

(8×2×8)×2=16×4×8×1.5=768 bits. And the buffer used to store the above one line of 4×4 blocks (AB) is (1920×4×8)+(960×2×8)×2=1920×4×8×1.5=92160 bits.

LB can be implemented by on-chip SRAM to improve performance, but obviously, AB is too big to be implemented totally by SRAM. As a tradeoff, we use on-chip SRAM of 6336 bits size as a cache to store the above 33 4×4 blocks, and a space in SDRAM is allocated to store one line of 4×4 blocks (chroma and luma samples are arranged side-by-side). Here, arranging cache size of 33 4×4 blocks is because of the special positions of blocks 23 and 15, thus the data transfer between cache and SDRAM happens at eight-MB intervals. Compared with the method to only use SDRAM for AB, this method spends identical time transferring data, but reduces the request overhead of SDRAM.

For intra prediction, H.264 uses the reconstructed but undeblocked MB data from the neighboring MBs to predict the current MB coefficients, and the used neighboring data are: (1) one sample of the above-left MB; (2) one line of samples of the above MB; (3) four samples of the above-right MB (only for luma components in intra 4×4 mode); (4) one column of samples of the left MB. We use on-chip SRAM to store the above one line of undeblocked data, and because of relatively small data structures of one column of data of the left MB, register is used to store them to speed up data access.

Estimation of SDRAM bandwidth

Considering the High-Speed Memory Bus as shown in Fig.3 is the performance bottleneck of the whole system, the highest work frequency of the hardware part is decided by the High-Speed Memory Bus. Because the SDRAM access performance of each module is unknown, at the beginning of this design, we can only approximately estimate the SDRAM bandwidth as shown in Table 1.

Output control module, which takes charge of playing the decoded video data, needs one Frame Volume. For bit stream input, the data and instruction access of RISC and other unconsidered demands, we arrange 2 Frame Volumes. According to the design requirement of 30 fps, the Bus bandwidth is 0.72~1.08 GBps. Under an 80% conservative estimate of SDRAM, the final SDRAM bandwidth is

Table 1 Estimation of SDRAM bandwidth

Module	Value*
Output control	1
Store	1
Motion compensation	4~8
Other	2
Total	8~12
Bus bandwidth	0.72~1.08 GBps

* Default unit: Frame Volume, which is the reconstructed pixel data (for 4:2:0 sub-sampling format) of one frame with the 1920×1088 resolution, namely about 3 MB

1.08/80%=1.35 GBps, thus the work frequency of the decoder is 166 MHz (64-bit SDRAM).

It is easily deduced that by the architectural modification of Section 3, the SDRAM bandwidth is reduced by 2/(12+2)~2/(8+2), namely 14%~20% compared with the method used in H.264 reference decoder.

IMPLEMENTATION RESULTS

In the development process of the decoder, we use FPGA prototype for design verification. The implementation results for the main modules of FPGA and ASIC are shown in Table 2.

Table 2 Implementation results of our decoder

Module	Resources used for the FPGA ($\times10^3$ LUT)	Resources used for the ASIC ($\times10^3$ Gate)
SDRAM controller	2.5	21
Deblocking&storing	11	93
Intra prediction	7	55
Inter prediction	8	64
Total	48	557

Among the modules, SDRAM Controller connects SDRAM to High-Speed Memory Bus. The FPGA is Xinlinx Virtex II 6000 and our chip has been synthesized in Simic 0.18 μm CMOS technology at 200 MHz. Considering the work frequency of 166 MHz of this decoder, the number of cycles needed to decode one MB is within $166\times10^6/(1920\times1088\times30\div256)=678$.

In this design, the simulation results of the decoder performance are shown in Table 3. On the average, for decoding I-slice, P-slice and B-slice MBs, deblocking&storing module takes comparatively steady cycles (78), while prediction module takes 156 cycles, 354 cycles and 546 cycles, respectively. Thus the requirement of 678 cycles needed per MB is satisfied.

Table 3 Average clock cycles needed per MB

Module	I-slice MB	P-slice MB	B-slice MB
Deblocking&storing	78	78	78
Prediction	156	354	546

CONCLUSION

In this paper, we proposed an MB-level decoding and deblocking method for H.264 decoder when FMO and ASO are used. The whole system is a system-on-chip (SOC) solution. In order to reduce the implementation difficulty and hardware implementation cost, the architecture is modified to decode MB in the raster scan order, which makes the decoder work on an MB basis, thus MB-level decoding and deblocking 3-stage pipeline is implemented in this design. On our FPGA verification platform, the highest work frequency is 50 MHz, and the decoding speed of video with 1920×1088 resolution is 6.7 fps when the work frequency is 32 MHz. Thus when the decoder works at 166 MHz, the decoding speed can reach 35 fps, which means this implementation can achieve the real-time decoding of 1080 HD (1920×1088@30 fps).

References

Chen, T.W., Huang, Y.W., Chen, T.C., Chen, Y.H., Tsai, C.Y., Chen, L.G., 2005. Architecture Design of H.264/AVC Decoder with Hybrid Task Pipelining for High Definition Videos. IEEE International Symposium on Circuits and Systems. Kobe, Japan, p.2931-2934.

Ha, V.H.S., Choi, S.K., Jeon, J.G., Lee, G.H., Jang, W.K., Shim, W.S., 2004. Real-Time Audio/Video Decoders for Digital Multimedia Broadcasting. The 4th IEEE International Workshop on System-on-Chip for Real-Time Applications. Banff, Alberta, Canada, p.162-167.

Iverson, V., McVeigh, J., Reese, B., 2004. Real-Time H.264-AVC Codec on Intel Architectures. IEEE International Conference on Image Processing. Singapore, p.757-760.

Kang, H.Y., Jeong, K.A., Bae, J.Y., Lee, Y.S., Lee, S.H., 2004. MPEG4 AVC/H.264 Decoder with Scalable Bus Architecture and Dual Memory Controller. Proceedings of the 2004 International Symposium on Circuits and Systems. Vancouver, Canada, p.II-145-148.

Khan, M.O., Khan, U., Rahim, S.A., Ali, S.I., 2004. Optimization of Motion Compensation for H.264 Decoder by Pre-calculation. The 8th IEEE International Multitopic Conference. Lahore, Pakistan, p.55-60.

Lee, J., Moon, S., Sung, W., 2004. H.264 Decoder Optimization Exploiting SIMD Instructions. IEEE Asia-Pacific Conference on Circuits and Systems. Tainan, Taiwan, p.1149-1152.

Lee, S.H., Park, J.H., Kim, S.W., Ko, S.J., Kim, S., 2006. Implementation of H.264/AVC Decoder for Mobile Video Applications. IEEE Asia and South Pacific Conference on Design Automation. Yokohama, Japan, p.120-121.

Park, S., Cho, H., Jung, H., Lee, D., 2005. An Implemented of H.264 Video Decoder Using Hardware and Software. IEEE Custom Integrated Circuits Conference. San Jose, CA, USA, p.271-275.

Ramadurai, V., Jinturkar, S., Moudgill, M., Glossner, J., 2005. Implementation of H.264 Decoder on Sandblaster DSP. IEEE International Conference on Multimedia and Expo. Amsterdam, Netherlands.

Wenger, S., Horowitz, M., 2002a. FMO: Flexible Macroblock Ordering, JVT-C089. Joint Video Team (JVT) 3rd Meeting. Virginia, USA.

Wenger, S., Horowitz, M., 2002b. FMO 101, JVT-D063. Joint Video Team (JVT) 4th Meeting. Klagenfurt, Austria.

Wiegand, T., Sullivan, G.J., Bjntegaard, G., Luthra, A., 2003. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, **13**(7):560-576. [doi:10.1109/TCSVT.2003.815165]