Signal Processing: *Image Communication* 24 (2009) 312–323

Contents lists available at ScienceDirect

Signal Processing: *Image Communication*

journal homepage: www.elsevier.com/locate/image

Platform-independent MB-based AVS video standard implementation

Xin Jin [a,b,*], Songnan Li [a], King Ngi Ngan [a]

[a] *Department of Electronic Engineering, The Chinese University of Hong Kong, Hong Kong, China*
[b] *Department of Electronics and Information, Huazhong University of Science and Technology, Wuhan 430074, China*

ARTICLE INFO

Article history:
Received 2 June 2008
Received in revised form
20 December 2008
Accepted 22 December 2008

Keywords:
AVS video standard
MB-based architecture
Frame-based architecture
Embedded video codec implementation
Video coding

ABSTRACT

AVS1-P2 is the newest video standard of Audio Video coding Standard (AVS) workgroup of China, which provides close performance to H.264/AVC main profile with lower complexity. In this paper, a platform-independent software package with macroblock-based (MB-based) architecture is proposed to facilitate AVS video standard implementation on embedded system. Compared with the frame-based architecture, which is commonly utilized for PC platform oriented video applications, the MB-based decoder performs all of the decoding processes, except the high-level syntax parsing, in a set of MB-based buffers with adequate size for saving the information of the current MB and the neighboring reference MBs to minimize the on-chip memory and to save the time consumed in on-chip/off-chip data transfer. By modifying the data flow and decoding hierarchy, simulating the data transfer between the on-chip memory and the off-chip memory, and modularizing the buffer definition and management for low-level decoding kernels, the MB-based system architecture provides over 80% reduction in on-chip memory compared to the frame-based architecture when decoding 720p sequences. The storage complexity is also analyzed by referencing the performance evaluation of the MB-based decoder. The MB-based decoder implementation provides an efficient reference to facilitate development of AVS applications on embedded system. The complexity analysis provides rough storage complexity requirements for AVS video standard implementation and optimization.

1. Introduction

AVS1-P2 is the newest video coding standard of Audio Video coding Standard (AVS) workgroup of China, which was initiated by the government of China in 2003 and was approved to be the national standard GB/T 200090.2-2006 in year 2006 [1]. AVS1-P2 is based on the hybrid coding framework including 8×8 block-based prediction, integer transform and entropy coding techniques. It aims at applications such as digital broadcast, high-density laser-digital storage media and etc. Currently, it is being applied in IPTV and China Mobile Multimedia Broadcasting (CMMB). AVS promises improved performance close to H.264/AVC main profile [2] with lower complexity [3].

Being a national video standard, AVS1-P2 has already drawn great attention from the industry. Generally, there are two categories of video decoder implementations: one is the software-based decoder implementations for embedded video codec or applications on PC platform; the other is the hardware-based decoder implementations for chip design in consumer electronic devices. Most of the literatures about AVS decoder implementations focus on the second category [4–11]. Some of them propose the hardware decoder architecture for AVS video decoder

* Corresponding author at: Waseda University, Japan.
E-mail addresses: goldcamel2004@yahoo.com, xjin@ee.cuhk.edu.hk (X. Jin), snli@ee.cuhk.edu.hk (S. Li), knngan@ee.cuhk.edu.hk (K.N. Ngan).

0923-5965/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.image.2008.12.003

implemented in a chip [4,5]; some of which just emphasize the implementation method for a specific coding tool, like variable length code (VLC) architecture for very-large-scale integration (VLSI) [6], the memory optimization for the VLC tables [7] by the data compression storage method, motion vector predictor architecture [8] for both AVS and MPEG-2, and VLSI implementation architectures for motion compensation (MC) [9–11].

It is a remarkable fact that the software-based decoder implementations are playing an equally important role in AVS industrialization, since there are so many applications suitable for software implementation especially as the general purpose CPU and Digital Signal Processor (DSP) becoming more and more powerful these days. However, literatures describing the software-based implementations of AVS video decoder are limited and they are focusing on the implementation method of individual decoding modules. The pipeline solutions to the interpolation module in MC and de-blocking filter are designed in [12,13], respectively, for DSP platform; code optimization work for I frame decoding is introduced in [14] for AVS video decoder on DSP. However, no paper compares the architecture differences for the software-based implementation from the point of view of application scenarios.

For the video applications on PC platform, decoder with frame-based architecture, similar to that of the reference decoder [15], is suitable to fully exploit the large memory resources on PC to pursue fast decoding speed. All of the reference information is saved for the whole frame; Intra prediction and de-blocking filter are directly performed in the reconstruction frame; motion compensation is performed directly in the reference frame. However, the frame-based architecture is not appropriate for embedded systems since the size of on-chip memory is too limited to save all of those frame information, and data transfer between the on-chip memory and the off-chip memory is time consuming [16]. Consequently, a platform-independent software package of AVS1-P2 decoder is proposed in this paper to facilitate AVS video decoder development in embedded system. Different from the frame-based architecture designed for PC-oriented applications, a macroblock-based (MB-based) decoding architecture is implemented to minimize the requirements in on-chip memory, to simulate the on-chip/off-chip data transfer, to modularize on-chip and off-chip processing, so as to facilitate embedded video codec implementation on the target platform. Compared to the frame-based architecture implementations, the MB-based architecture reduces over 80% on-chip memory requirements in decoding 720p sequences. Since the MB-based solution is platform-independent, it can be easily transferred to many kinds of platforms as DSP or system on chip (SoC) by platform-dependent optimization and modules reordering.

The rest of this paper is organized as follows. After a brief introduction of AVS1-P2 standard in Section 2, a platform-independent MB-based software package is proposed for embedded video codec development in consumer electronic devices in Section 3. Its system architecture is presented with a comparison with the frame-based decoder architecture. The detailed implementation methods of the MB-based decoder, as buffer update, Intra prediction, motion compensation and de-blocking filter, are described in Section 4. The storage complexity of the MB-based implementation is measured through the comparison with the frame-based architecture in Section 5. The efficiency in saving the on-chip memory by using the MB-based architecture is proved, and the storage requirements for decoding videos with different resolutions are provided. Finally, conclusions are presented in Section 6.

2. AVS1-P2 standard overview

Like other popular video coding standards, the decoding process of AVS1-P2 is fully specified in the standard. The decoding process consists of the bitstream parsing and data processing to generate a reconstructed video sequence. A hybrid block-based video codec is defined for AVS1-P2, which is similar to H.264 [2] and H.263 [17].

The general diagram of AVS1-P2 decoder is depicted in Fig. 1. Targeting the HD broadcasting and storage applications, AVS1-P2 is designed based on hierarchical bitstream structure from the high-level syntax, such as sequence header and picture header, down to the slice level data and macroblock (MB) level data. The number of MBs within a slice must be an integral multiple of the image width divided by the MB width. A luminance MB consists of 16×16 samples, while an MB of each of the two chrominance components consists of 8×8 samples for 4:2:0 sample format. After the high-level syntax is decoded and stored, a simple adaptive two-dimensional

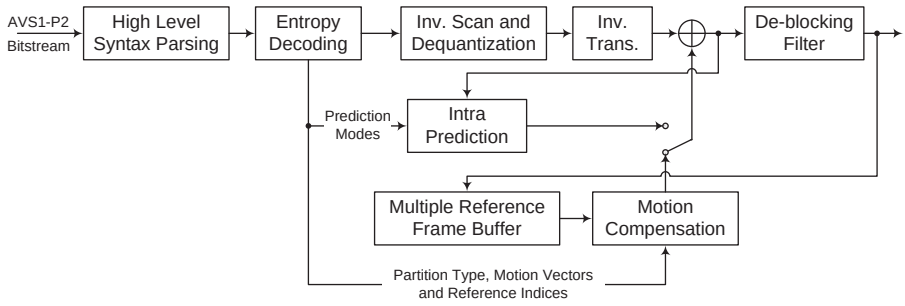

Fig. 1. AVS1-P2 decoding diagram.

variable length code (2D-VLC) is defined as the entropy decoding method to interpret the Golomb codes into the level-run pair for the quantized residuals of each 8×8 block. Whether to run the 2D-VLC is determined by the coded block pattern (CBP). After entropy decoding, 8×8 inverse scan, dequantization followed by 8×8 inverse integer transform are performed for multiple times to reconstruct a residual MB.

As defined by the coding mode of each macroblock, either Inter prediction or Intra prediction can be used to generate the prediction of a macroblock. For Inter prediction, four partition modes listed as 16×16, 16×8, 8×16 and 8×8 are enabled for motion compensation together with the maximum two reference frames and quarter-pixel interpolation accuracy. If a motion vector points to a sub-sampled position, two four-tap interpolation filters are used in generating the luminance pixel in half-sample position and quarter-sample position, respectively; while, a bilinear filter is used to interpolate the chrominance reference sample with eighth-pixel accuracy.

For Intra prediction, five 8×8 directional modes, including vertical, horizontal, DC, down left and down right modes, are defined for luminance and four modes, including DC, horizontal, vertical and plane modes, are defined for chrominance. The reference samples involved in Intra prediction are those decoded and reconstructed neighboring samples. For luminance, maximum 16 samples above the current block, 16 samples left to the current block and one sample up-left to the current block are used in interpolation. For chrominance, maximum 9 samples above, 9 samples left to and one sample up-left to the current block will be used in generating the prediction.

After generating the predicted MB and the residual MB, the two MBs are added together to form the final reconstructed MB. Finally, an in-loop de-blocking filter is applied to reduce the blocking artifacts in the reconstructed image to improve the subjective quality of the output and the objective quality of MC. Since filtering is based on the 8×8 block edges of both the luminance and chrominance, a maximum of four edges, two horizontal and two vertical, will be filtered for a luminance MB and two edges will be filtered for each chrominance MB. Three levels of filtering strength from 0 to 2 are defined which will be determined by the MB coding type, motion vector and reference index. For the filtering strength other than zero, the type of filter and the number of samples that will be filtered are determined by the pixel difference and edge thresholds. Up to a maximum of three samples on one side of an edge will be filtered. After the whole image is filtered, the reconstructed image will be put into the buffer that saves the reference frames for Inter prediction and be sent to the output after reordering, if needed, for display or storage.

3. System architecture

3.1. Frame-based AVS decoder

For video playback applications on PC platform, the basic requirement is to provide real-time decoding speed under different system statuses for a variety of video resolutions. Since a video buffer control is generally used in such video playback application to guarantee a constant video display speed, the requirement for the software-based video decoder core is to pursue a fast decoding speed. In order to let the program work efficiently on PC platform, the program architecture needs to fully exploit the hardware resources and to fit for the features of the applications. Compared to the embedded system, PC usually has much more powerful microprocessor, high-speed cache, wide bandwidth bus, large physical memory and tremendous sized hard disk. While, compared to video encoding process, video decoding is a kind of memory-intensive application, which requires frequent memory access, data processing and transfer. Consequently, fully exploiting the memory resource is a significant consideration in designing the system architecture for AVS1-P2 video decoder on PC platform. The frame-based system architecture for PC-oriented AVS1-P2 decoding applications is depicted in Fig. 2.

As shown in the figure, the decoder is built on two hierarchies. In order to decode a frame, the decoder first calls the functions to decode the high-level syntax in the bitstream like sequence header (if it is the first frame), picture header, slice data and etc. After the sequence header is decoded, the frame-based buffers are initialized to save the sequence information and MB information for the whole frame. The sequence information includes the image resolution parsed from the sequence header, and the frame type and coding parameters that interpreted from the picture header. The MB information includes MB type, Inter or Intra prediction mode, reference index, motion vector (MV) difference, CBP, QP delta and etc., which will be retrieved during slice data decoding. As it starts to decode slice data, the decoder triggers the function Decode_one_macroblock() repeatedly until the entire slice of data is decoded. Decode_one_macroblock() is the major decoding function for an MB which consists of several low-level kernels like macroblock overhead decoding, entropy decoding, Intra prediction, MC and etc. All of these kernels are constructed for processing a macroblock of data at a time. After the overhead of a macroblock is decoded, the MB information is saved into those frame-based buffers mentioned above. Intra prediction is performed directly in the reconstructed frame buffer, and the sub-pixel interpolation and motion compensation are performed directly in the reference frame buffer. The output of function Decode_one_macroblock() is a reconstructed MB, which will be put into the reconstructed frame buffer. After the entire frame is decoded, an in-loop de-blocking filter is applied on the reconstructed frame to remove the blocking artifacts. If the filtered reconstructed frame is a reference frame, it will be put into the reference frame buffer for motion compensation. For the storage application, the decoded bitstream will be written to the hard disk or other storage devices. For the real-time playback applications, the decoded frame will be put into a jitter buffer to provide a constant display speed.

The frame-based architecture efficiently exploits the system resources of PC platform. The frame-based architecture is designed mainly considering the resource

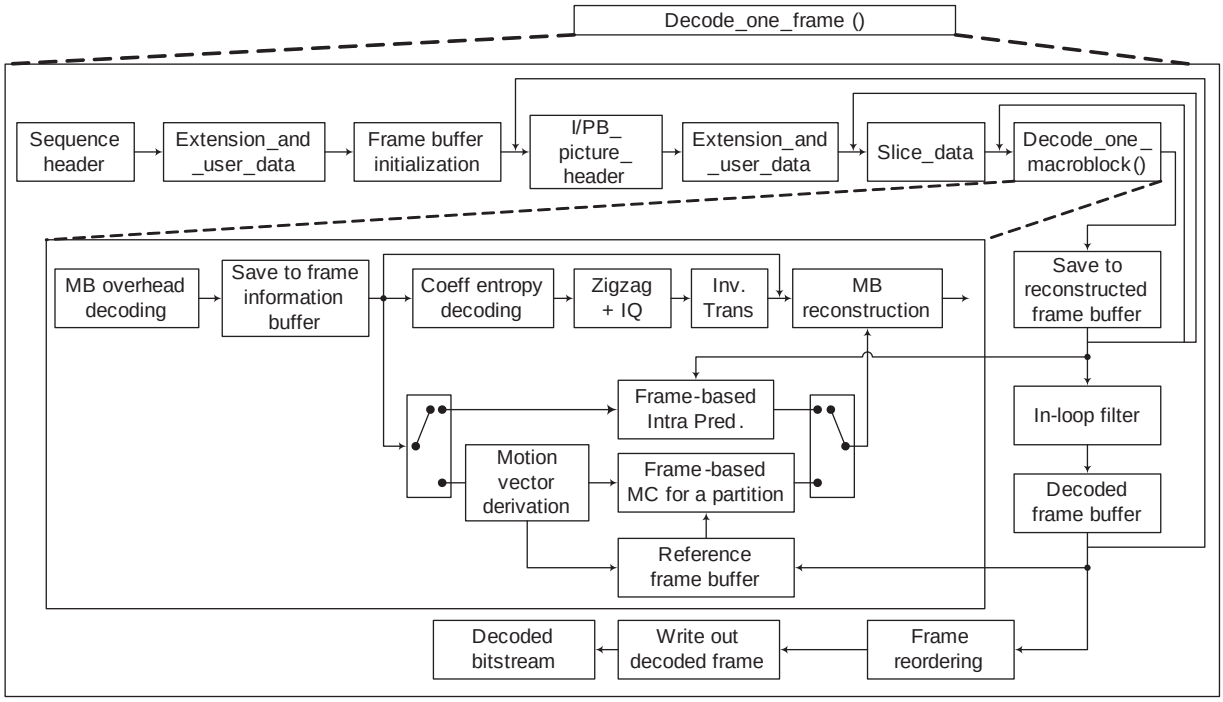

Fig. 2. Frame-based AVS decoder diagram and code partition.

and bandwidth performance of the storage devices on PC. Since the size of physical memory in PC is very large, generally more than 1 gigabytes, all of the information of a frame can be saved in a buffer to avoid frequent information update and judgment; Intra prediction can be performed directly in the reconstructed frame buffer and the de-blocking filter is performed in the final reconstructed frame buffer to efficiently save the time consumed in data access, moving and update. Since the bandwidth performance for PC storage devices can be roughly ordered from low to high as hard disk, physical memory and CPU cached memory [18], and the cached memory provides a pre-fetching mechanism, saving the temporary data in the memory with successive address can increase the cache hit ratio so that to save the time consumed in data transfer between the high-speed memory and low-speed memory.

3.2. MB-based AVS decoder

Compared to the PC platform, an embedded system encapsulates a complete computing system for a specific application, which will be embedded into a larger system later. Though the embedded systems are application-specific, they still show some common attributes. Generally, they are designed to meet some constraints in cost, power consumption and design time [19]. Considering the cost, the on-chip memory in those systems is generally less than 1 megabytes [16,20], which is much smaller than the PC memory. Though the off-chip memory is relative large, the data transfer between the on-chip memory and the off-chip memory is time and power consuming [19]. Consequently, the frame-based architecture is not appropriate for this application. Considering the diversity of applications of embedded system, a platform-independent software package is proposed and developed to save the development time in embedded codec design for AVS1-P2 decoder. Data transfer flow is simulated, the MB-based system architecture is designed and the major decoding sub-functions are modularized to facilitate embedded video codec implementation on the target platform.

Fig. 3 depicts the data flow used in the proposed MB-based decoder. Since this decoder is developed for embedded system, the data flow of the software package simulates the data communication in the target platform. The compressed bitstream file is stored on the hard disk. After the file is loaded to the off-chip memory (as SDRAM on PC), the decoder loads the bitstream piece by piece from the off-chip memory to the on-chip bitstream buffer via the dedicated on-chip/off-chip data transfer routines. After decoding, the reconstructed MBs are written from on-chip to off-chip to form the reconstructed frame. Via the file input/output interface, the reconstructed frame is written to hard disk to save as decoded sequence.

Based on the above data flow, the architecture of the MB-based decoder is depicted in Fig. 4. Compared to the frame-based architecture shown in Fig. 2, most of the high-level syntax parsing modules are the same except that no global frame-sized buffers are initialized after the sequence header is decoded. As the slice data starts to be decoded, a function called MB_decode_one_macroblock()

is triggered repeatedly to decode the entire slice data. Compared to the function Decode_one_macroblock() in the frame-based architecture, there are three major features possessed by MB_decode_one_macroblock(). Firstly, a set of MB-based buffers is used during the whole decoding process. After MB overhead, such as MB type and Inter or Intra prediction information, is decoded, a process denoted as "Update ref. buffer" is executed to update the information in a set of MB-based buffers by saving the current MB information and the neighboring reference information. If the current MB is the first MB of the decoded sequence, "Update ref. buffer" performs initialization for these MB-based buffers. Secondly, MB_decode_one_macroblock() consists not only of overhead decoding, entropy decoding, Intra prediction and MC like Decode_one_macroblock(), but also of loop-filter and decoded MB output. All of these kernels are constructed for processing a macroblock of data at a time. There are dedicated routines for data transfer between on-chip and off-chip (e.g. loading the bitstream and reference partitions, and writing out the reconstructed MBs). Thirdly, all of the decoding kernels in MB_decode_one_macroblock() are performed in the MB-based buffers described above, which only save the current MB information and the neighboring reference information that needed by the current MB decoding.

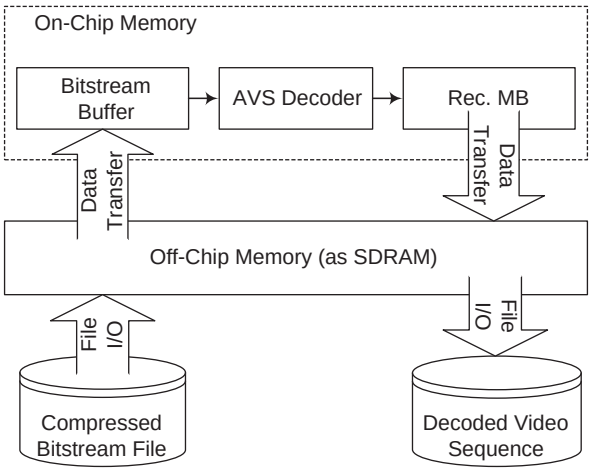

Fig. 3. MB-based AVS decoder data flow.

In Section 4, the implementation of four major MB-based modules denoted as MB-based buffer update, Intra prediction, MC and de-blocking filter will be described in detail with the emphasis on buffer definition, data management and update process, while the decoding algorithm of Intra prediction, MC and de-blocking filter will not be described since they are the same with that defined in the standard [1].

4. MB-based AVS decoder implementation

In this Section, the detailed implementation methods of some key decoding kernels are described for the MB-based decoder with the emphasis on buffer definition and data management process.

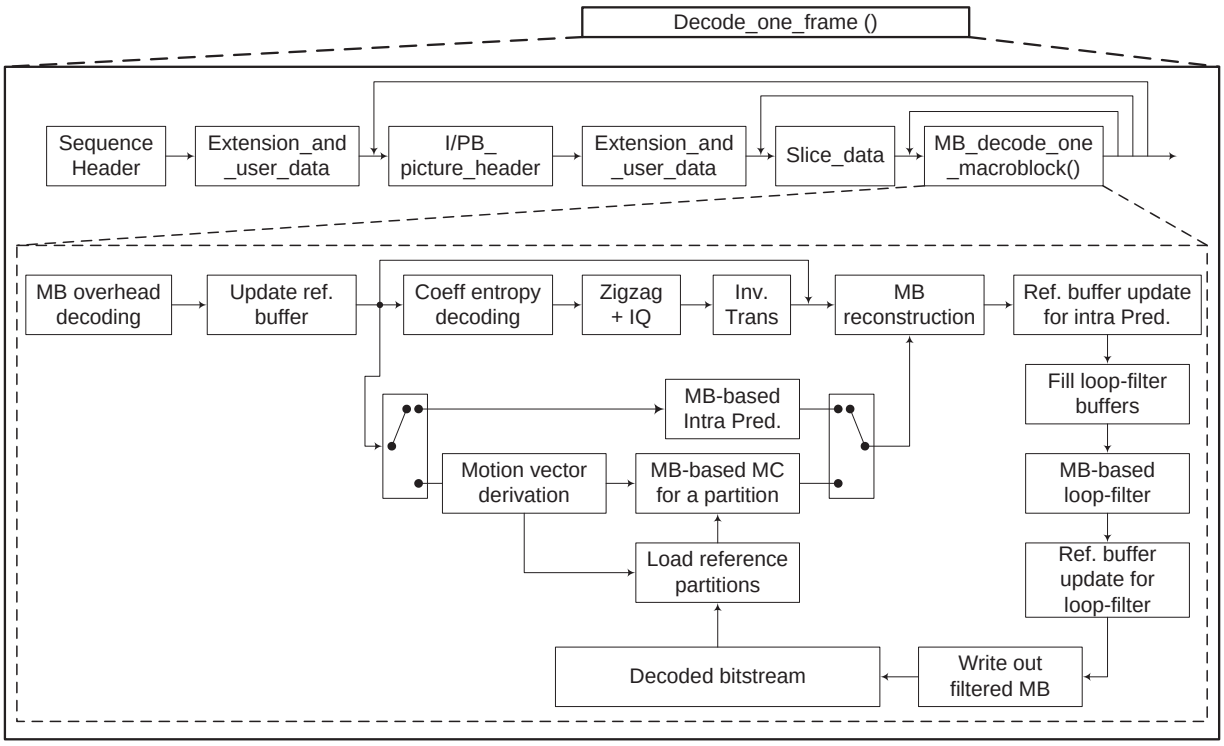

Fig. 4. MB-based AVS decoder diagram and code partition.

4.1. MB-based buffer update

In order to minimize the on-chip memory, most of the low-level decoding kernels are processed in the MB-based buffers which save the information only needed for decoding the current MB. Before decoding the next MB, these buffers need to be updated by saving the just decoded MB's information for following decoding and by replacing the neighboring information for the next MB. This buffer update process is denoted as "Update ref. buffer" in Fig. 4. Here, four sets of decoding information buffers defined for MV, reference index (RI), QP and luma Intra prediction mode (LIPM) are introduced.

Using the buffers for MV as an instance, Fig. 5 illustrates the buffer definition and update method. *Up_Ref_MotionVector* is the reference buffer saving all of the decoded MVs of the whole block line above the current MB. *MotionVector_Pred_Buffer* is the working buffer for MV-related decision, of which X_0–X_3 save MV of each block of the current MB, others save prior decoded MVs of spatially neighboring blocks. If the neighboring MB has partition type other than 8×8, the MV of larger partition will be duplicated to each 8×8 block for saving. Before decoding the next MB, MB numbered as n+1, these two buffers need to be updated to save the just decoded MVs of MB n, and to refresh the neighboring MVs for MB n+1. The update consists of three steps: first, the MVs in U_3, X_1 and X_3 are copied to the left of the current MB as indicated by arrow 1 to update the left reference MV; then, MVs in r_{2n} and r_{2n+1} in *Up_Ref_MotionVector* are replaced by values in X_2 and X_3 for decoding the MVs of the vertically adjacent MB; thirdly, MVs in r_{2n+2}, r_{2n+3} and r_{2n+4} in *Up_Ref_MotionVector* are copied to U_2, U_3 and *UR* above the current blocks in *MotionVector_Pred_Buffer*. Using such buffer update, the MVs saved in the working buffer are ready for decoding MB n+1, and the MVs in *Up_Ref_MotionVector* are automatically updated for decoding the next MB line.

Similar to motion vector, the reference index uses the same size of reference buffer and working buffer. The working buffer for QP is only 2×2, since each MB has only one QP value and the QP of the MB above-right of the current MB is not needed. The size of reference QP buffer is image width divided by the width of an MB. For the buffers used to save LIPM, a 3×3 working buffer and an image_width/block_width+2 reference buffer are used. Values saved in the first and the last elements of reference buffer are -1, which represent the image boundary. The updating methods for all of these buffers are similar to that of MV.

4.2. MB-based Intra prediction

Different from the frame-based implementations that perform Intra prediction directly in the reconstructed frame, the MB-based luma Intra prediction is implemented in the three buffers as shown in Fig. 6. The 16×16 buffer with bias is a common buffer for decoding the current MB. It saves the predicted samples during Intra prediction or Inter prediction and updates them to be reconstructed samples by adding the prediction value with the reconstructed residue after inverse quantization and inverse transform. *Intra_upbuf_Y* saves the whole line of decoded samples above the current MB. *Intra_lefbuf_Y* saves the decoded samples left to the current MB, of which the first element, denoted as *C*, is the sample spatially above-left to the current MB, and the 17th element *Intra_lefbuf_Y* [17] equals to the 16th.

The buffer access during luma Intra prediction and buffer update method are interpreted in Fig. 7. As depicted in the figure, current intra-coded MB is divided into four 8×8 blocks X_0–X_3, which will be decoded sequentially. Since each intra-coded 8×8 block needs to use prior decoded samples in adjacent blocks to generate prediction block, different blocks uses samples in different buffers as

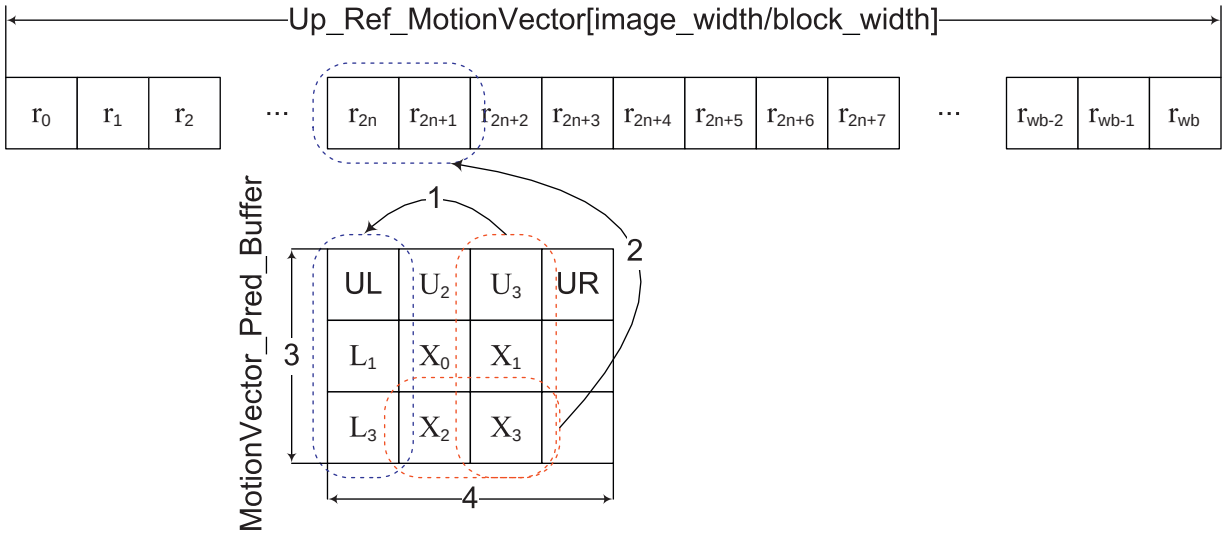

Fig. 5. Buffer definition and update for motion vectors.

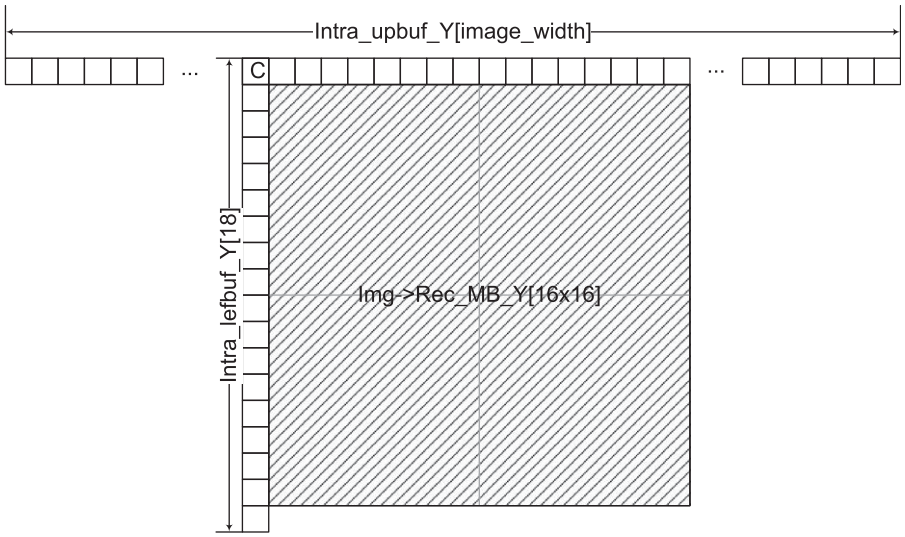

Fig. 6. Luma Intra prediction buffer definition.

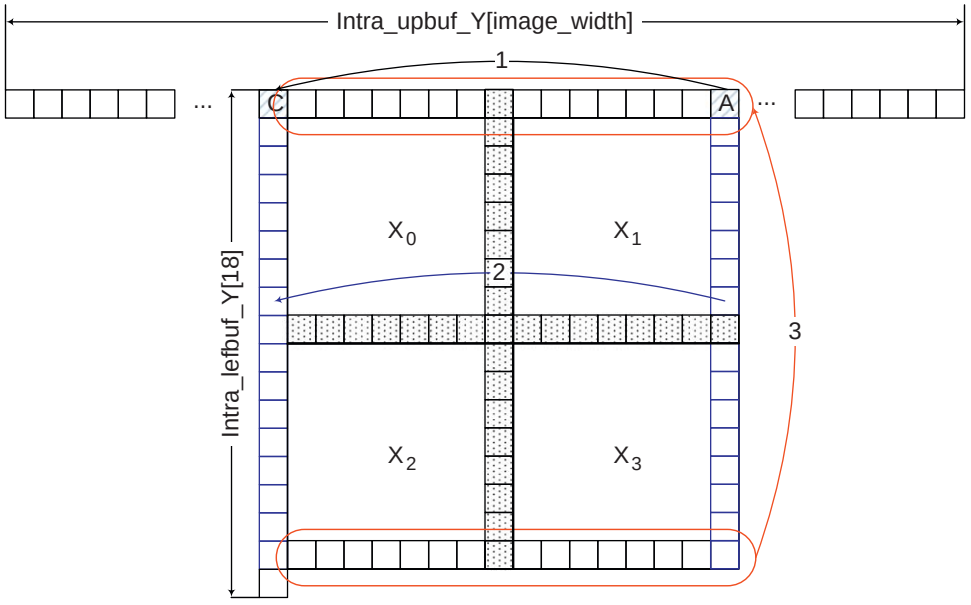

Fig. 7. Reference pixel allocation and update for luma Intra prediction.

its reference. For X_0, samples in *Intra_upbuf_Y* and *Intra_lefbuf_Y* are its reference samples; for X_1, the right column of samples in X_0 (8 dotted samples) and *Intra_upbuf_Y* are its reference; for X_2, the samples in *Intra_lefbuf_Y* and the bottom row in X_0 (8 dotted samples) are its reference; for X_3, the dotted right column in X_2 and the dotted bottom row in X_1 are its reference.

After finishing decoding the whole MB, no matter the MB is intra-coded or inter-coded, the pixel buffers will be updated as: copy pixel *A* in *Intra_upbuf_Y* to position *C* in *Intra_lefbuf_Y* following arrow 1; copy the right column of samples of the current reconstructed MB (16 samples in blue) to the corresponding position in *Intra_lefbuf_Y* as indicated by arrow 2; thirdly, copy the bottom row of samples (16 samples in red circle) from the current MB buffer to the corresponding position in *Intra_upbuf_Y* following arrow 3. Using such mechanism, *Intra_lefbuf_Y* is ready for decoding the next MB, and *Intra_upbuf_Y* is automatically ready for decoding the next MB line after the current MB line is decoded.

The buffer update for chroma (4:2:0) Intra prediction is almost the same with luma Intra prediction. The only difference is the buffer size defined for it. The size of the buffer saving the samples above the current MB is half of the image width. The size of the buffer saving the samples left to the current MB is 10.

4.3. MB-based motion compensation

Motion compensation in the decoder is used to retrieve Inter predicted partition according to the motion information. For the frame-based implementation, interpolation is directly performed in the reference frame. However, the size of an entire reference frame is too large for to be loaded into the on-chip memory for an embedded system. Consequently, the MB-based MC is working as depicted in Fig. 8.

As shown in the figure, MC first retrieves the reference partition with integer-pixel accuracy from the reference frame according to the motion information and loads it to the on-chip memory via data transfer routines; then the interpolation is performed in on-chip memory to generate predicted partition with higher accuracy. Since the interpolation algorithm is the same with that described in the standard, only the reference partition loading method will be introduced in the following.

Still using luma as an instance, how many integer samples need to be loaded from the reference frame is determined by the partition size and where the motion vector points to. The size of partition can be one of four partition sizes defined as 16×16, 16×8, 8×16 and 8×8. While, the pointed position of motion vector is more complex. In case the motion vector points to an integer-sample position (denoted as *A* in Fig. 9), the same number of integer samples as the partition size needs to be loaded into the on-chip memory; otherwise, a larger number of integer samples needs to be loaded to generate those noninteger positions. Table 1 lists the number of integer samples that needs to be loaded into the on-chip memory according to the MV positions depicted in Fig. 9.

As shown in the table, in case the motion vector points to half-sample position as "1", partition with (*partition_width*+3) × *partition_height* integer samples needs to be loaded into on-chip memory. Based on such loading method, the maximum on-chip buffer size used to save the reference partition is 20 by 20, which is much smaller than the size of an entire reference frame, and the on-chip/off-chip data transfer is well modularized for target platform.

The reference partition loading flow is much easier for chroma (4:2:0), since its interpolation method is much simpler than luma [1]. Only (*partition_width*+1) × (*partition _height*+1) integer samples need to be loaded into the on-chip memory if MV does not point to an integer-sample position.

4.4. MB-based de-blocking filter

Different from those frame-based implementations that filter MBs after the whole image is reconstructed,

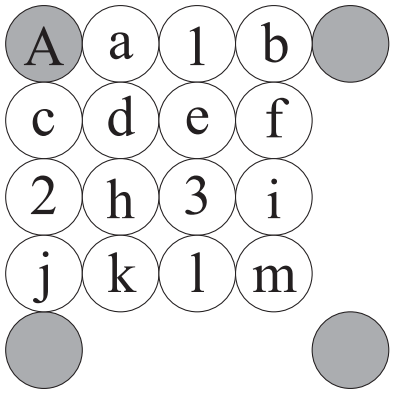

Fig. 9. Interpolation positions: gray samples are samples at integer-position; samples numbering 1–3 are samples at half-sample position; others are samples at quarter-sample position.

Table 1
The number of integer reference samples according to different MV positions.

MV position	The number of integer samples ($W\times H$)	
	W (Width)	H (Height)
A	partition_width	partition_height
1	partition_width+3	partition_height
2	partition_width	partition_height+3
3	partition_width+3	partition_height+3
a, b	partition_width+4	partition_height
c, j	partition_width	partition_height+4
h, i	partition_width+4	partition_height+3
e, l	partition_width+3	partition_height+4
d, f, k, m	partition_width+3	partition_height+3

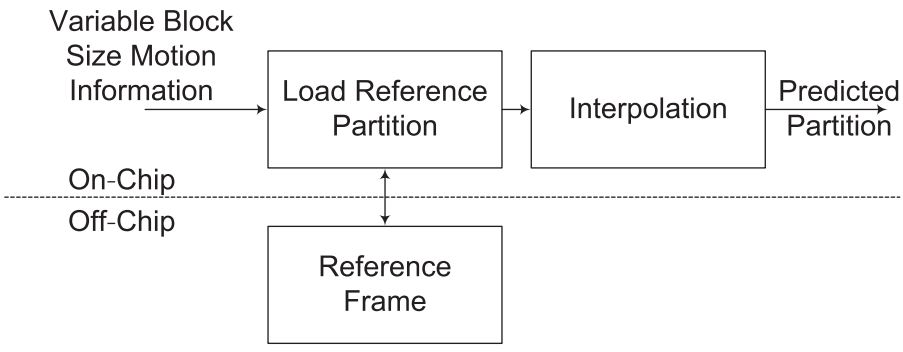

Fig. 8. MB-based motion compensation flow.

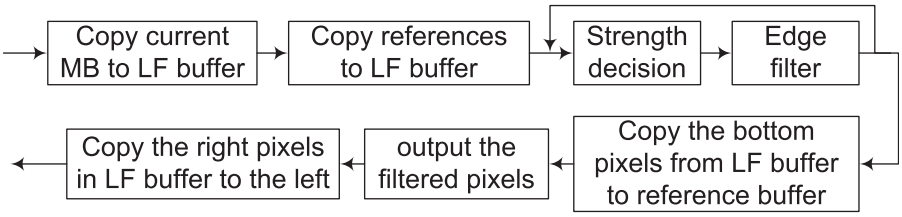

Fig. 10. MB-based de-blocking filter flow.

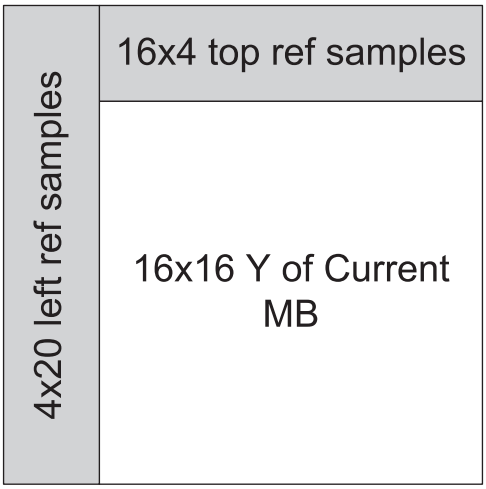

Fig. 11. Loop filter working buffer for luma.

MB-based de-blocking filter filters each MB immediately after it is reconstructed so that to save the on-chip memory and memory bandwidth. The modules related to the MB-based filtering are shown in Fig. 10.

As shown in the figure, the strength decision and edge filter are the standard defined filtering processes, which will not be introduced in this paper. While, the processes before and after those two modules are designed specially for the MB-based data allocation and buffer update, which will be described in detail in the following using luma as an instance. There are two MB-based buffers defined for luma de-blocking filter: one is the major filter working buffer, saving the reconstructed MB and the reference samples left to and above the current MB; the other is reference buffer saving prior filtered lines of samples above the current MB. The size of the working buffer is 20×20 as shown Fig. 11. The size of reference buffer is image width by 4.

Based on the above two buffers, the buffer management flow before filtering is illustrated in Fig. 12. As shown in the figure, the reconstructed MB is firstly copied to the bottom-right corner of the loop-filter buffer as indicated by arrow 1. Then, the reference samples spatially above the current MB are copied from the reference buffer into the working buffer as indicated by arrow 2.

After filtering four edges of the current MB, the data copy and buffer update scheme is shown in Fig. 13. The bottom-left 16×4 samples (area with gray bias) are first copied to the reference buffer to replace those samples that were copied in step2 in Fig. 12 for filtering vertically adjacent MB. The top-left 16×16 final filtered pixels (gray area) are secondly written out to the decoded frame via on-chip/off-chip data transfer routines. Finally, the 4×20 right samples (area with blue bias) are copied to the left as indicated by arrow 3 to be the reference samples for the next adjacent MB's filtering, during which the value of the samples will be further changed.

The buffer management and update manner of chroma (4:2:0) is almost the same with that of luma except the buffer size. The size of working buffer of any chroma component is 12×12, while the reference buffer size is a half of image width by 4.

5. Applications and complexity analysis

5.1. Applications

Based on implementation and modularization described in Section 4, it is easy to put our MB-based AVS decoder into a target platform. If the objective is to implement an AVS decoder on DSP, the platform-dependent instruction optimization needs to be performed for the software package to fully take advantage of the parallelism provided by the processing units. Some of the data transfer and control, including the bitstream loading, reference partition loading, reference frame and reconstructed frame output, need to be implemented by the direct memory access (DMA) engine, resulting in further improvement of the decoding speed due to the fact that CPU and DMA can work simultaneously.

If the objective is to design an AVS decoder chip using this MB-based decoder, it will be a little bit more complicated than for DSP applications. The platform-dependent instruction optimization also needs to be performed first. Then, the coding time of each module needs to be evaluated to reorder the modules to fully exploit the pipeline structure for parallelism. The parallelism structure is determined by the chip architecture. The correctness of each modification can be verified by using the proposed MB-based decoder as a good reference.

5.2. Complexity analysis

Since the MB-based decoder not only implements the decoding flow but also simulates data transfer between the off-chip memory and on-chip memory, and the computational complexity is highly platform dependant,

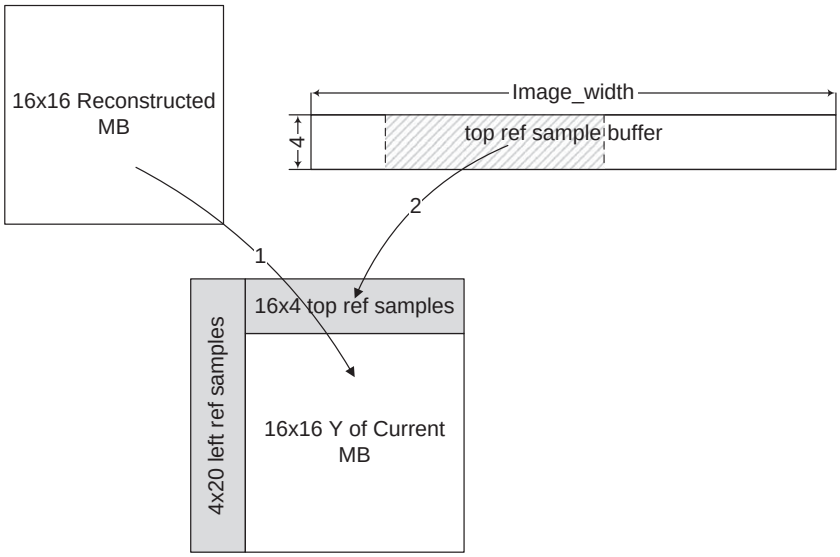

Fig. 12. Memory management before filtering.

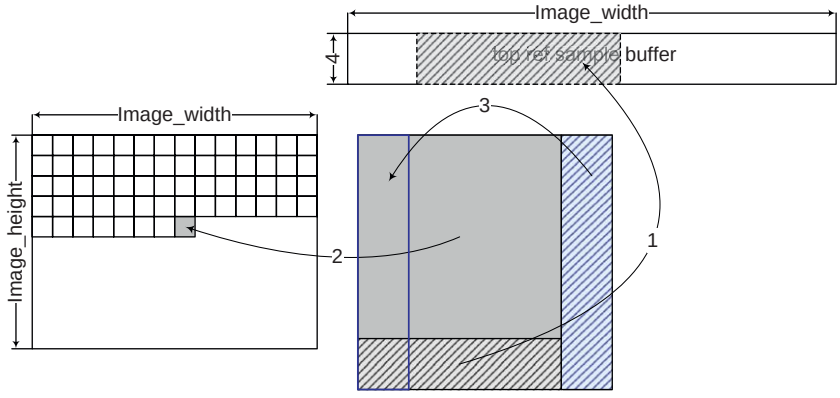

Fig. 13. Memory management after filtering.

only the storage complexity is analyzed in this section for the MB-based implementation by comparing with the frame-based one.

Table 2 presents a comparison of the storage requirements between the MB-based AVS decoder and the frame-based AVS decoder. As shown in the table, w and h represent the image width and height, respectively; n is the number of reference frames with the maximum value of 2. Since the frame-based decoder performs MC directly in the reference frame, the buffer used to load the reference partition is not available (denoted as N.A. in Table 2). De-blocking filter in the frame-based decoder is performed directly in the reconstructed frame after all of the MBs of the image have been decoded. If designing a chip using the MB-based decoder, the first four buffers in Table 2 will be put into off-chip memory, while others will be put into on-chip memory to avoid on-chip/off-chip data transfer. While, if using the frame-based decoder to design the chip, reconstruction frame and reference frames are required to be put into on-chip memory, which is extremely too expensive.

Table 2
Storage requirements, in bytes, for the MB-based AVS decoder and the frame-based AVS decoder.

Buffer name	Formula in the frame-based decoder	Formula in the MB-based decoder
Reference indices of previous frame	$w/8 \times h/8$	$w/8 \times h/8$
Motion vectors of previous frame	$w/8 \times h/8 \times 2 \times 2$	$w/8 \times h/8 \times 2 \times 2$
Reference frame	$n \times w \times h \times 1.5$	$n \times w \times h \times 1.5$
Reconstruction frame	$w \times h \times 1.5$	$w \times h \times 1.5$
Reference indices	$w/8 \times h/8 \times 2$	$(w/8+2)+4 \times 3 \times 2$
Motion vectors	$w/8 \times h/8 \times 2 \times 2 \times 2$	$((w/8+2) \times 4+$ $4 \times 3 \times 4) \times 2$
Reconstruction MB	$16 \times 16 \times 1.5$	$16 \times 16 \times 1.5$
Reference partition	N.A.	$20 \times 20+9 \times 9 \times 2$
QP values	$w/16 \times h/16$	$w/16+4$
Intra prediction mode	$(w/8+2) \times (h/8+2)$	$(w/8+2)+9$
Intra prediction	$16 \times 16+40 \times 3$	$(w+18)+(w/2+10) \times 2$
Deblocking filter	In reconstruction frame	$(w \times 4+20 \times 20)+(w/$ $2 \times 4+12 \times 12) \times 2$
MB temp data	~1566	~1565
Static tables	3961	3961

Table 3
Storage requirements, in bytes, for different image resolutions.

Buffer type		CIF, $w = 352$, $h = 288$	4CIF, $w = 704$, $h = 576$	720p, $w = 1280$, $h = 720$
Reference frame		152064	608256	1382400
Reconstruction frame		152064	608256	1382400
On-chip buffer	Frame-based decoder	24271	77891	168791
	MB-based decoder	11445	15537	30073
	MB-based vs. frame-based	1:2.12	1:5.01	1:5.61

Table 3 compares the buffer size for input videos with different resolutions as n equal to 1. The first two buffers listed in Table 2 are ignored in Table 3, since those two buffers must be put in the off-chip memory. As shown in the table, "On-chip buffer" refers to the latter 10 buffers listed in Table 2. For the frame-based decoder, the calculation regards the size of "Reference partition" and "De-blocking filter" as zero. Even though the size of reference frame and reconstruction frame are not taken into account, the MB-based decoder still saves significant amount of on-chip memory. The saving ratio is increased with the increment in image resolution. The MB-based buffer sizes listed in Tables 2 and 3 also provide an overall storage requirement for AVS1-P2.

Roughly comparing the decoding speed of the MB-based decoder and the frame-based decoder, if the on-chip memory of the chip is as large as the size of RAM used in current PC, the frame-based decoder must work faster than the MB-based decoder because of the time saving in buffer update and management. Unfortunately, current generic on-chip memory is less than 1 M bytes [20], where the MB-based decoder will work faster than the frame-based decoder because of the time saving in on-chip/off-chip data transfer.

6. Conclusions

In this paper, a platform-independent AVS decoder is developed for embedded video codec applications. Simulating the data access and transfer routines between the on-chip memory and off-chip memory, an MB-based architecture built on two hierarchies is developed for the decoder to perform all of the low-level decoding processes in a set of MB-based buffers with adequate definition, data allocation and buffer update. Compared to the frame-based architecture, which is commonly used for video playback applications on PC platform, the MB-based architecture can significantly save the on-chip memory. The modularization of function and data flow can also facilitate the AVS video decoder development on system on chip (SoC). Compared with the frame-based architecture, the MB-based solution can save over 80% of on-chip memory for 720p video sequences. The complexity analysis also provides a rough storage requirement for AVS video standard implementation.

Acknowledgements

The work was supported in part by the Hong Kong Applied Science and Technology Research Institute (ASTRI) and the Chinese University of Hong Kong Focused Investment Scheme (Project 1903003).

References

[1] AVS Video Expert Group, Information Technology-Advanced Audio Video Coding Standard Part 2: Video, in Audio Video Coding Standard Group of China (AVS), Doc.AVS-N1063, December, 2003.
[2] Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264/ISO/IEC 14 496-10 AVC), March 2003.
[3] Lu Yu, Feng Li, et al., Overview of AVS-Video: tools, performance and complexity, in: Proc. of SPIE International Conference on VCIP. Beijing, China, July 2005, pp. 679–690.
[4] C. Peng, C. Huang, R. Wang, J. Dai, Y. Zhao, Architecture of AVS hardware decoding system, in: Proceedings of International Symposium on Intelligent Multimedia, Video and Speech Processing, October 2004, pp. 306–309.
[5] Huizhu Jia, Peng Zhang, Don Xie, Wen Gao, An AVS HDTV video decoder architecture employing efficient HW/SW partitioning, IEEE Transactions on Consumer Electronics 52 (4) (2006) 1447–1453.
[6] Bin Sheng, Wen Gao, Don Xie, Di Wu, An efficient VLSI architecture of VLD for AVS HDTV decoder, IEEE Transactions on Consumer Electronics 52 (2) (2006) 696–701.
[7] Ke Zhang, Xiaoyang Wu, Lu Yu, An area-efficient VLSI implementation of CA-2D-VLC decoder for AVS, IEEE International Symposium on Circuits and Systems, New Orleans, LA, May 2007, pp. 3151–3154.
[8] Junhao Zheng, Di Wu, Lei Deng, Don Xie, Wen Gao, A motion vector predictor architecture for AVS and MPEG-2 HDTV decoder, in: Seventh Pacific-Rim Conference on Multimedia (PCM 2006), Hangzhou, China, 2006, pp. 424–431.
[9] Lei Deng, Wen Gao, Ming-Zeng Hu, Zhen-Zhou Ji, An efficient VLSI implementation for MC interpolation of AVS standard, in: Advances in Multimedia Information Processing-PCM 2004: fifth Pacific Rim Conference on Multimedia, Tokyo, Japan, 2004, pp. 200–206.
[10] Junhao Zheng, Lei Deng, Peng Zhang, Don Xie, An efficient VLSI architecture for motion compensation of AVS HDTV decoder, Journal of Computer Science & Technology 21 (3) (2006) 370–377.
[11] Dajiang Zhou, Peilin Liu, A hardware-efficient dual-standard VLSI architecture for MC interpolation in AVS and H.264, IEEE International Symposium on Circuits and Systems, New Orleans, LA, 2007, pp. 2910–2913.
[12] Shu Hu, Xiaoxia Zhang, Zhigang Yang, Efficient implementation of interpolation for AVS, Congress on Image and Signal Processing (CISP '08) 3 (2008) 133–138.
[13] Zhigang Yang, Wen Gao, Yan Liu, Debin Zhao, DSP implementation of deblocking filter for AVS, in: The 14th IEEE International Conference on Image Processing, San Antonio, Texas, 2007, pp. 205–208.
[14] Ban Wang, Jianyang Zhou and Meifen Chen, Implementation and optimization of I-frame in AVS1-P2 decoder based on DSP, in: International Conference on Communications, Circuits and Systems (ICCCAS 2008), 2008, pp. 754–757.

[15] AVS reference software, "rm52j_r1.zip," 〈FTP://159.226.42.57/incoming/dropbox/video_software/P2_software〉.
[16] Nikil D. Dutt, Alexandru Nicolau, On-chip vs. Off-chip memory: the data partitioning problem in embedded processor-based systems, ACM Transactions on Design Automation of Electronic Systems 5 (2000) 682–704.
[17] ITU-T Recommendation H.263, Video coding for low bit rate communication, 1998.
[18] Paul Hsieh, Programming optimization: techniques, examples and discussion, 〈http://www.azillionmonkeys.com/qed/optimize.html〉.
[19] Frank Vahid, Tony D. Givargis, Embedded system design: a unified harware/sofrware introduction, Wiley Inc., New York, 2001.
[20] DaVinci™ Digital Media Processors, 〈http://focus.ti.com/paramsearch/docs/parametricsearch.tsp?family=dsp&sectionId=2&tabId=1852&familyId=1300〉.