A High Efficient Memory Architecture for H.264/AVC Motion Compensation

Chunshu Li，Kai Huang, and Xiaolang Yan
Institute of VLSI Design
Zhejiang University
Hangzhou, China
Email:lics@vlsi.zju.edu.cn

Jiong Feng, De Ma, and Haitong Ge
Soc Design Group
C-sky Microsystem Corporation
Hangzhou, China
Email:jiong_feng@c-sky.com

***Abstract*—In H.264/AVC decoding system, motion compensation operation occupies about 80% of the total memory access and becomes the system bottleneck. In this paper, a high efficient memory architecture for H.264/AVC motion compensation is proposed to extremely reduce external memory access bandwidth. A four-level hierarchical memory organization scheme is utilized to explore the reusability of neighboring blocks at an acceptable area cost. To improve the system processing throughput, five optimization techniques are adopted in motion compensation operation, which enable video decoder to achieve real-time decoding of HD 1080p video stream when operating at 110 MHz. Compared with the existing works, the proposed architecture is able to reduce the memory bandwidth requirement in motion compensation progress by 83.7% and performs better in the real-time application.**

Keywords-H.264/AVC; Motion Compensation; memory architecture; VLSI

I. INTRODUCTION

H.264/AVC is the latest video compression standard which outperforms the earlier standards through a series of aggressive compression techniques. However, in order to support these techniques, huge memory bandwidth is required to achieve real-time decoding through the dedicated VLSI implementation. The motion compensation (MC) operation occupies about 80% among the memory access required in the decoding system [7]. Above 90% of the memory access in MC operation is caused by fetching the reference data during sub-pixel interpolation. Owing to the long latency of access to DRAM, too much DRAM access affects decoding performance greatly.

There are two popular strategies utilized to reduce reference data fetching times in current H.264 MC hardware designs. One is to avoid reading redundant reference pixels according to the interpolation position (AIP) [3]. During sub-pixel interpolation, all the 81 reference pixels should be loaded in typical H.264/AVC reference software (JM), which brings much redundant memory access. For instance of the interpolation positions a/b/c defined in [1], it requires only the shaded region shown in Fig. 1. About 55% unnecessary reference pixels are loaded. Another strategy is to explore the spatial locality of the reference pixels and reuse the shared reference data by adjacent blocks (RSR)[2]-[8]. This strategy makes use of on-chip SRAM or register file to cache those data with high reusability. Using on-chip SRAM or register file is able to reduce DRAM access while increases chip area cost.

Furthermore, on-chip memory may bring extra DRAM access if memory architecture is not organized well. Therefore, an efficient MC memory architecture is necessary to make full use of on-chip memory to reduce DRAM access, making an acceptable tradeoff between video decoder performance and chip area cost.

II. RELATED WORK

Recently four typical researches based on AIP and RSR strategies were undertaken to improve the efficiency of MC memory access.

Ke Xu [4] proposed a balanced design between system throughput and power consumption at the cost of a buffer using 169 (13x13) bytes. The proposed technique achieves a reduction of memory access by avoiding reading the repetitive data shared by adjacent 4x4 blocks. But the data reusability for 4x4 blocks are constrained in one 8x8 region and it can not exploit the data reusability of adjacent 8x8 blocks and macroblocks (MB). According to our experimental results, the 8x8 blocks and MBs who share reference pixels with their neighbors occupy about 90% and 30% of all the 8x8 blocks and MBs considered respectively, which illustrates there is a great need to exploit the reusability. Furthermore, in the worst case, the data path of pixel interpolation has to wait for 52 cycles before the reference pixels of an 8x8 block are all read back, which leads to low performance.

Yu Lei [5] used an overlapped region duplication (ORD) technique to avoid reading overlapped reference pixels across 8x8 block level boundaries, which can improve the system performance a lot. However, there are four deficiencies in this design: 1) Two register buffers, using 338(2x13x13) bytes in all, are used to implement the ORD technique, which causes high hardware cost in the chip implementation; 2) All the reference data fetching from the

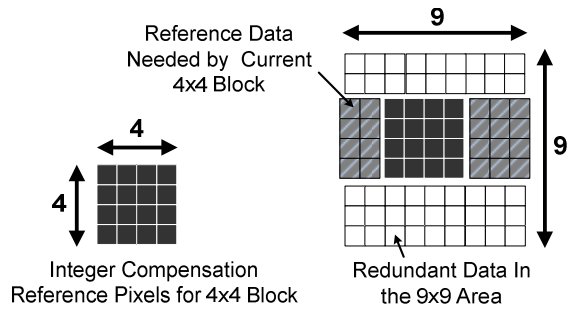


Figure 1. The reference data needed by the interpolation positions "a/b/c"

external memory have to be written to both of the buffers, which brings large power consumption; 3) It requires a (M+5)x(N+5) reference block for each MxN partitioned block for luminance component, which may load many useless reference pixels. It degrades system performance and causes unnecessary power consumption; 4) It can not reuse the shared reference data by adjacent MBs.

A cache-based strategy was described in [6] and [7]. This cache-based scheme can reuse the reference data shared by adjacent blocks of different level. However, the cache schemes in [6] aim at software implementation, which are not fit to hardware implementation. Moreover, when cache prefetch fails, the performance is reduced for the reason that many cycles are wasted on the useless data loading. Design [7] proposed a novel cache scheme to reduce MC bandwidth requirement by more than 70% which achieved remarkable improvement compared to other designs. However, the size of cache used in [7] is up to 1536 (16x16x6) bytes. Besides, additional 1632-bit (16x6x17) registers are used for implementing the cache scheme. Considering on-chip memory and registers occupy a large portion of the total chip area, the area cost of the proposed scheme in [7] is somewhat high.

Overall, two main problems can not be well solved in the previous works: 1) The shared reference data by adjacent 8x8 blocks and MBs can not be reused efficiently at a low hardware cost; 2) The general-purpose cache prefetch and replace scheme used in the previous works can not work efficiently for the application-specific MC hardware design.

Based on the AIP and RSR strategies, the proposed memory architecture in this paper adopts a four-level hierarchical memory architecture, namely IMC, IMB, IRB and IIR, to explore the reusability for the shared reference data at different levels: 4x4 block, 8x8 block and MB. Moreover, in order to accelerate the throughput of MC, a fine memory organization on IRB is used to adapt to MC pipeline stages. A pipeline scheme to 4x4 block's reference pixels' fetching and interpolation macro stages (FIP) is adopted. This scheme avoids the decoding system's long wait until the 8x8 block's reference pixels are all read back. The proposed architecture can reduce the external memory bandwidth of MC by 83.7% and can efficiently solve the excessive external memory access problem.

Thanks to the efficient buffer organization scheme, the proposed memory architecture uses a local register buffer size up to 137(9x13+4x5) bytes and an on-chip memory size up to 6k bits. Compared with the similar cache-based design in [7], the local buffer requirement is reduced by about 50%.

The rest of the paper is organized as follows. Section III describes the detailed hierarchical memory organization scheme mentioned above. Section IV shows the implementation results. Finally section V concludes the paper.

III. PROPOSED MEMORY ARCHITECTURE

The following sections explain the four-level memory scheme presented in Fig. 2 in detail.

The memory architecture is composed of level 1 IIR for intermediate results saving during interpolation operation, level 2 IRB for reference data reuse in one 8x8 block, level 3 IMB for reference data reuse of 8x8 blocks and neighboring MBs, and level 4 IMC for reference data reuse in one frame.

A. Intermediate Interpolation Registers (IIR)

In the proposed memory architecture, the IIR acts as the first-level memory to reuse the intermediate results in a 4x4 block's interpolation process, so as to cooperate with the parallel interpolation process to speed up the throughput of the interpolation engine.

For implementing the 2-D filtering operation, there are three kinds of filters utilized in the interpolation engine: horizontal 6-tap filters, vertical 6-tap filters and average filters. The interpolation process is shown in Fig. 3(b) using the interpolation position "i" as example. To reuse the shared operands by vertical and average filters, the intermediate interpolation results generated from their pre-stage filters are saved into IIR as shown in Fig. 3(a).

Take consideration of the interpolation position "i" whose interpolation progress can be found in H.264/AVC standard[1], the horizontal interpolation results in cycles 1, 3, 5, 7 and the vertical interpolation results in cycles from 1 to 8 will be saved into IIR.

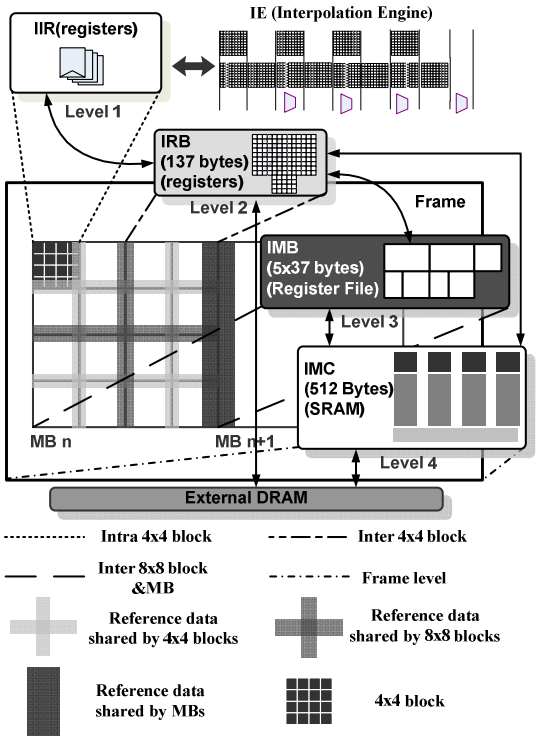


Figure 2. Four level memory hierarchy

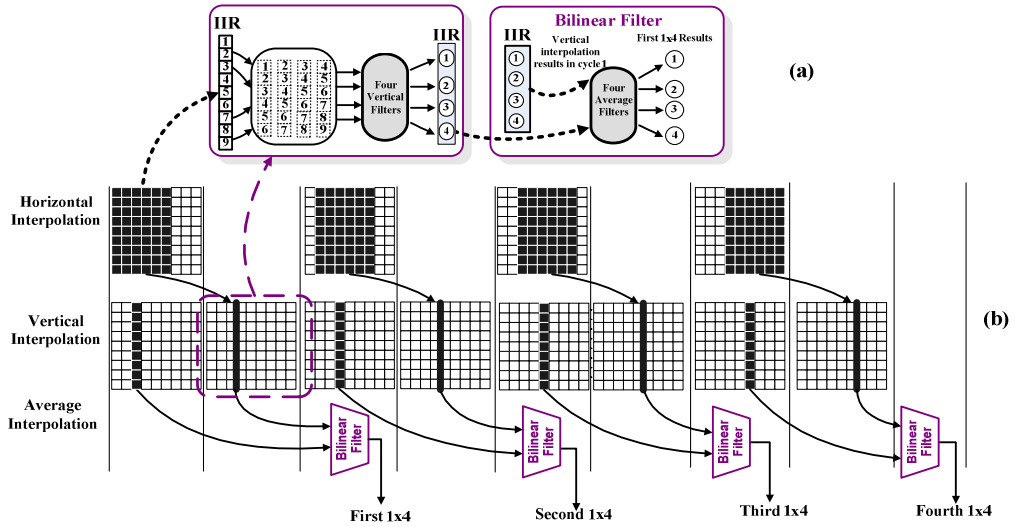


Figure 3. Interpolation process for position "i"

B. Internal Register Buffer (IRB)

A buffer using 137 bytes acts as the second level which deals with the four 4x4 blocks' decoding process inside an 8x8 block. A fine memory organization scheme is used to adapt to the FIP scheme which accelerates the system throughput by 25.3% and reduces the buffer size by 19% compared to [4].

For the decoding process of one 4x4 block, 81 (9x9) reference pixels are required in the worst case. Thus, for an 8x8 block, up to 324 reference pixels are necessary to be read at most. So much DRAM access brings a heavy burden on the off-chip memory bandwidth and leads to rapidly increasing power consumption as well. However, when the four 4x4 blocks inside an 8x8 block share one motion vector, only 169 reference pixels are required for the whole 8x8 block's interpolation with reuse of the shared reference data by the internal four 4x4 blocks. The total memory access times can be reduced by 47.8%.

According to our experimental result shown in Fig. 4, most of the 8x8 blocks (more than 86%) use the same motion information for the internal four 4x4 blocks. Therefore it is quite necessary to utilize IRB to manage the reuse of reference pixels for the adjacent 4x4 blocks in one 8x8 block.

In our design, a register buffer using 137 bytes is used to meet the requirement of an entire 8x8 block's decoding process with the help of a fine memory organization scheme. If the partition block size is smaller than 8x8, the four 4x4 blocks are dealt with separately. The storage organization of the reference pixels required by each 4x4 block is shown in Fig. 5.

The shaded region stands for the reference data needing to be loaded whose shape varies with the interpolation positions.

When the four 4x4 blocks in one 8x8 block region share one motion vector, a pipeline scheme for the four 4x4 blocks' reference pixels' fetching and interpolation macro stages (FIP) is adopted as shown in Fig. 6.

This pipeline scheme avoids the long wait until all the reference pixels of an 8x8 block are ready and improves the

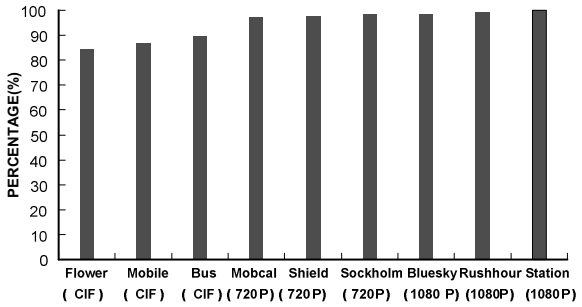


Figure 4. Percentage of 8x8 blocks who share the same motion information inside

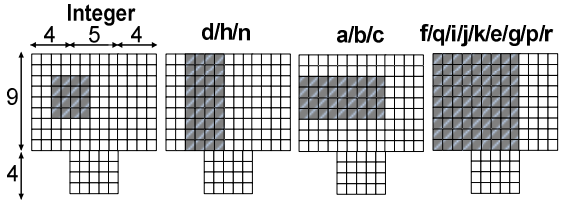


Figure 5. Storage organization in IRB of reference pixels required by a 4x4 block

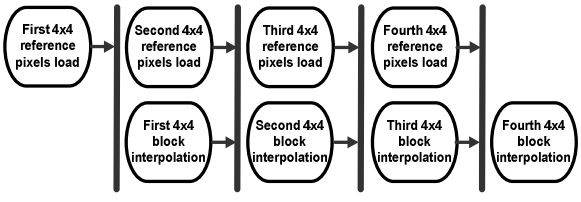


Figure 6. Fetching and interpolation pipeline scheme for four 4x4 blocks inside an 8x8 block

system throughput compared to [4]. Table I illustrates the storage organization of the reference pixels required by the four 4x4 blocks in an 8x8 block in IRB. Load sequence in Table I describes the loading sequence of all the reference pixels required by the four 4x4 blocks. Storage

organization describes how to store the reference pixels in each loading step. For example, for d, h and n interpolation positions, both region 1 and region 3 marked in load sequence share region 5 marked in storage organization. Meanwhile, both region 2 and region 4 marked in load sequence share region 6 marked in storage organization. The similar organization scheme adapts to the situation of f, q, i, j, k, e, g, p and r interpolation positions. This organization simplifies the hardware logic of selecting source operands for the following interpolation engine and does not cause any conflict between blocks 1 and 3 or 2 and 4.

C. *Internal Macro-block Buffer (IMB)*

IMB, implemented with a 185-byte register file, acts as the third level which stores the shared reference data by adjacent 8x8 blocks and MBs. Fig. 7 (a) and (b) illustrate the shared reference data region by adjacent 8x8 blocks and MBs respectively. As mentioned above, when all the 4x4 blocks in one 8x8 block share one motion information, 169 (13x13) reference pixels are required to read back from DRAM at most. However, when its adjacent 8x8 blocks to the left or top also have the same motion information, 45 (5x9) reference pixels can be shared which is shown in Fig. 7 (a) and the corresponding DRAM can be cut out. When it comes to the adjacent MBs situation, the figures are 441 (21x21) and 105 (21x5) respectively which is shown in Fig.7 (b).

The experiment result about the amount of shared reference pixels by adjacent 8x8 blocks and MBs is shown in Fig. 8. The percentage of the amount of 8x8 blocks who share reference data with their neighbors is about 76.4%~99.9%. Moreover, the analysis shows that the percentage is further increased as the frame resolution size is enlarged. It also shows a great opportunity to utilize the shared reference data by adjacent 8x8 blocks. As for adjacent MBs, the percentage of the amount of MBs who share reference data with their left neighbors is about 30%.

TABLE I. STORAGE ORGANIZATION IN IRB FOR FIP SCHEME

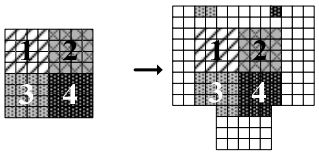
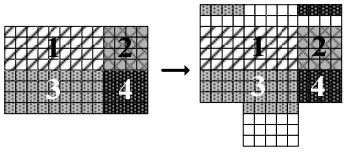
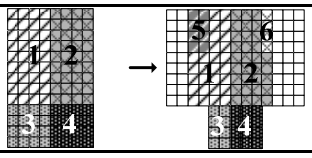
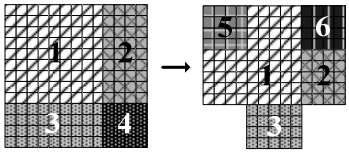
Position	Load sequence	Storage organization
integer		
a/b/c		
d/h/n		
f/q/i/j/k /e/g/p/r		

Fig. 7 (c) shows all the reference data required for one MB when all the sixteen 4x4 blocks inside share one motion vector. The shaded region represents the shared reference data by adjacent 8x8 blocks and MBs and is stored in IMB.

A buffer using 265 bytes is required to store all the reference data in shaded region. To reduce hardware cost, we propose a buffer strategy which utilizes a 185-byte buffer to implement the reusability as shown in Fig. 7 (d).

In the first 8x8 block decoding process, the shared regions 1, 2 and 3 shown in Fig. 7 (c) are loaded from the external DRAM or IMC and stored into IMB as shown in Fig. 9 (a). The regions 5 and 6 stored into IMB during the last MB's decoding process are exported to the internal register buffer (IRB) for the interpolation computation. In the second 8x8 block decoding process, the regions 4, 5` and 6` are stored into IMB as shown in Fig. 9 (b). The regions 2 and 3 stored into IMB during the first 8x8 block's decoding process are exported to IRB. A similar memory organization scheme can be used for the third and fourth 8x8 blocks' decoding process. The IMB regions in and out of the dotted line work by turns for each MB's decoding. For non-shared regions a, b and c, they are loaded from IMC or SDRAM and stored into IRB directly each time they are required.

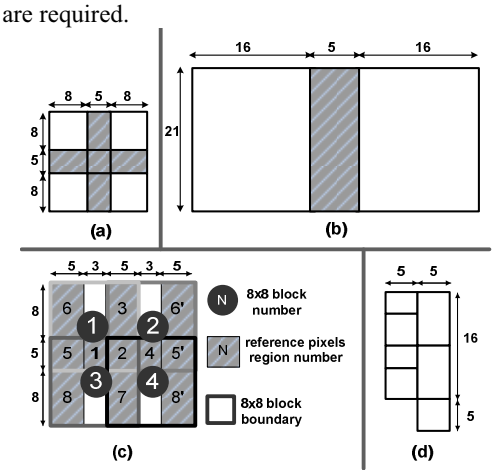


Figure 7. IMB buffer illustration

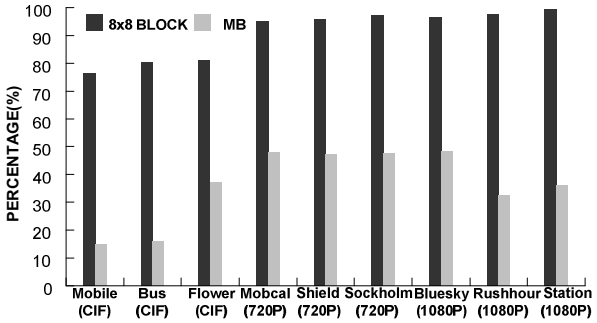


Figure 8. Analysis of the amount of shared reference pixels by adjacent 8x8 blocks and MBs

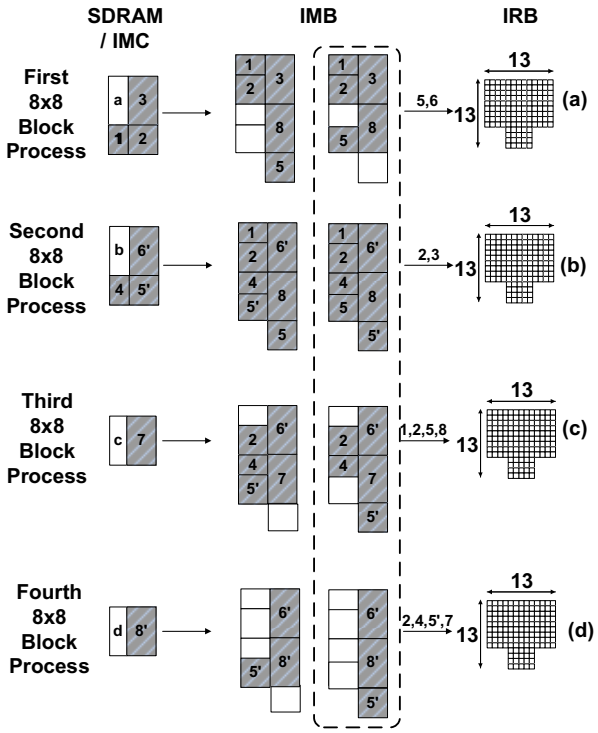


Figure 9. The IMB organization process

D. Internal Memory Cache (IMC)

To further explore the data reusability, an internal memory cache mechanism is utilized to store those reference pixels recently used. Experimental result in [10] indicates that the value of y component of motion vector is mostly less than 2. Therefore there is a high possibility for the current block to reuse the reference data of those adjacent blocks with the same MVY value. Furthermore, it's also found the value of x component of motion vector is mostly distributed in the range [-8, 8]. It indicates that, in most cases, current block shares reference data with other blocks which are not too far.

The proposed IMC uses a four-way cache architecture to get a better tradeoff between the performance improvement and the area cost. Fig. 10 shows the experimental results about how much DRAM access is reduced when different IMC scheme is used. Four-way Cache is more acceptable owing to the almost 50% DRAM access reduction without too much area cost.

The IMC memory map is shown in Fig. 11: each line of data array is organized as 8 pixels in one 4x4 block. Each

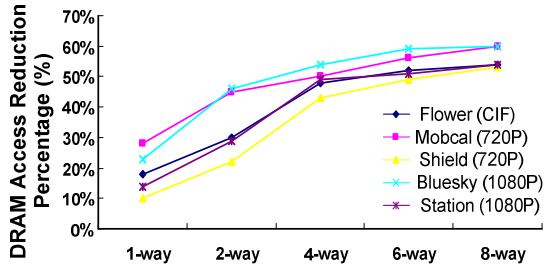


Figure 10. DRAM access reduction for different cache scheme

16x16 block is divided into two banks: bank0 and bank1. Each bank consists of 8 sub-banks and each sub-bank involves two lines.

The tag array of IMC is composed of three parts: RI, MBy and MBx. RI is a 4-bit reference frame index of the reference block. MBx and MBy are X and Y components of MB address respectively. This array is implemented by four separated banks of register file (each bank's size is 144 bits). Each line of tag array bank has two bits: V bit for indicating current line valid and BA bit for choosing bank0 or bank1. SBA is the last 7 bits of the memory access address and is used for line selection. When an access to IMC arrives, according to its SBA, four banks of tag array will check the validation and BA correctness of the corresponding tag line. Either not valid or BA wrong occurs, the hit signal of this tag bank will not be asserted. Otherwise, Tag Array is enabled and the corresponding tag is selected to compare with RI, MBy and MBx of current access address. Finally the tag_hit signal is generated.

Cache address generator can generate data array address with four input tag_hit signals. If there is not any tag_hit signal valid, this cache access is missed and the reference pixels have to be read from external DRAM. Meanwhile the corresponding cache line is refilled. Data array of IMC is implemented as one 512-byte SRAM.

IV. RESULTS AND COMPARISON

The hardware architecture of H.264/AVC video decoder design is shown in Fig. 13, involving multiple computation units, interconnection network and memory subsystem. The data communication of the whole system is based on bus matrix for high throughput. One multi-layer DMA and on-chip memory subsystem can make use of the bus matrix to improve the parallelism of data transfer. The main computation kernel consists of one main MB sequencer control unit and four computation units which include MC unit being shaded.

A usual Network Abstraction Layer unit (NALU) is composed of sequence parameter set (SPS), one picture parameter set (PPS) and coded slices. PPS, SPS and the

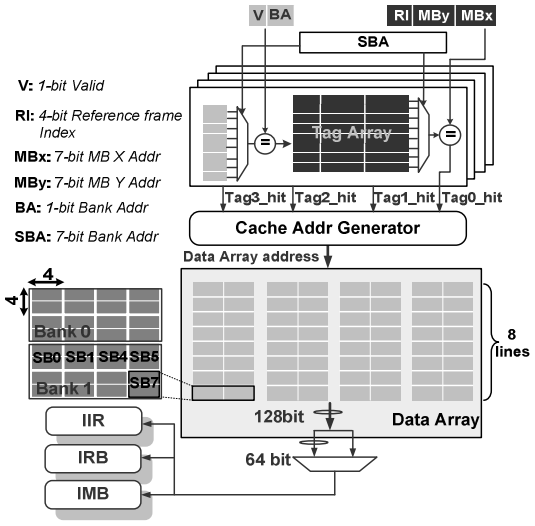


Figure 11. IMC architecture

header of coded NALU slice will be processed directly by CPU main system. Then CPU designed by Hangzhou C-sky Microsystem Corporation stores sequence parameters, picture parameters and slice parameters into registers or memory which video decoder accelerator can access. After that, CPU configures and starts video decoder for slice data decoding. When one NALU slice data decoding is completed, video decoder generates an interrupt to CPU for requesting the permission to decode next NALU slice data.

The proposed hardware architecture was implemented by Verilog Hardware Description Language (HDL) and synthesized by Design Compiler with SMIC 65nm Silicon Process.

For the system throughput performance measurement, the dedicated testing sequences generated by JM 8.6 encoder were used to verify the target design. From the results of our experiment, the proposed design can achieve real-time decoding of HD 1080p (1920x1088 @30fps) video stream when operating at 110 MHz. The throughput is greatly improved by five techniques in our design. Fig. 12 illustrates the contribution of four techniques, AIP, IRB, IMB and IMC, to the final performance improvement. As we can see, without any optimization, the average number of DRAM access times for one MB decoding is about 528 times. Four optimization strategies, AIP, IRB, IMB and IMC, contribute 28.8%, 32.3%, 32.3% and 49.6% to the memory bandwidth requirement reduction respectively. By use of these four optimization strategies, the average number of DRAM access times is finally reduced to about 86 times per MB which reduces the memory bandwidth requirement by 83.7%. Moreover, the FIP technique accelerates the system performance by paralleling the reference data loading and interpolation process. The above five optimization techniques enable our design to decode each MB within 500 cycles.

The chip cost and throughput performance are compared with several buffer-based or cache-based designs in references in Table II .

From the comparison results shown in Table II, in terms of gate count, our proposed design outperforms designs [4], [7] and [9] by 11.3%, 27.3% and 43.0% respectively.

In terms of local memory cost, our design outperforms design [7] by 50.0%. Although the local memory size is larger than that of design [4], it brings about 30.7% throughput improvements.

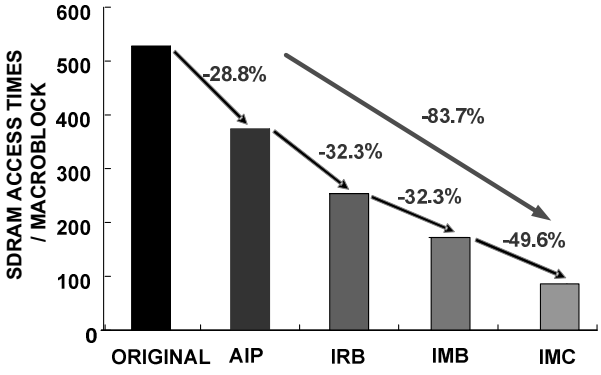


Figure 12. Analysis of system throughput improvement four techniques contribute

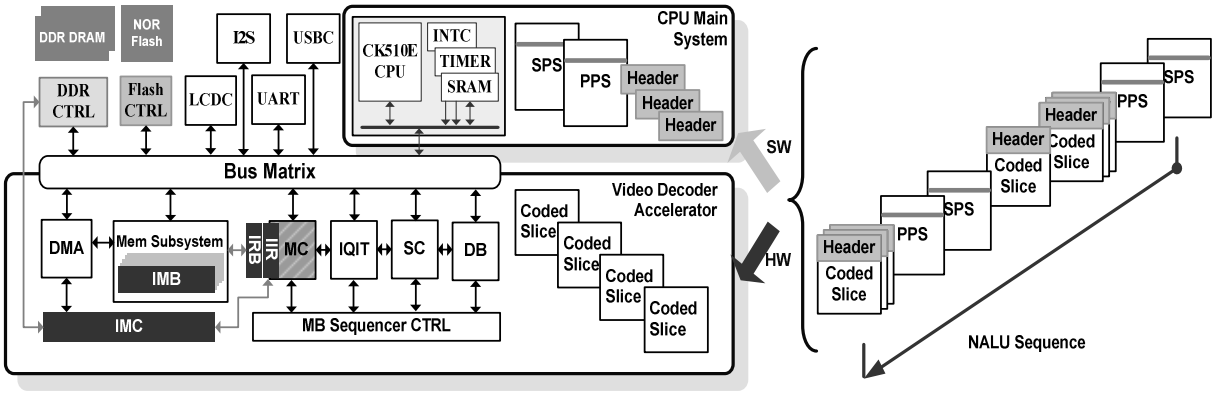


Figure 13. Block diagram of H.264/AVC video decoder with the proposed MC memory architecture

TABLE II. COMPARISON OF PROPOSED DESIGN WITH OTHER WORKS

	Proposed Design	**Design [4]**	**Design [5]**	**Design [7]**	**Design [9]**
Logic Gates	53k	60.6k	NA	72.3k	93k
Local Memory	RF: 2.056K bits SRAM: 4k bits	0 bits	NA	12k bits	NA
Performance	1080p@30fps 110MHz	QCIF@30fps 1.5MHz	D1@30fps 55MHz	NA	CIF@30fps 20MHz
Reduced DRAM Access Times/MB	83.7%	53%	NA	71.6%	NA
Average Cycles/MB	456**	500*	650*	NA	850*

*: Without considering DRAM accessing latency **: With three-cycle DRAM accessing latency

In terms of the throughput performance, our design outperforms the designs [4] and [7] by 30.7% and 12.1% in measurement of average DRAM access times for one MB decoding, and outperforms the design [9] by 46.4% in measurement of the average cycles required in one MB decoding process.

V. CONCLUSION

This paper proposes a four-level hierarchical memory organization scheme for MC operation in H.264/AVC decoding process. Through five optimization techniques, namely AIP, IRB, IMB, IMC and FIP, the implemented decoder can achieve real-time decoding of HD 1080p (1920x1088@30fps) video stream when operating at 110 MHz which outperforms the existing H.264 video decoders listed in Table II. The proposed MC architecture was implemented by SMIC 65nm Silicon Process technology and used 53k gates and 6k-bit on-chip memory, achieving a better tradeoff between the performance and the area cost.

REFERENCES

[1] Joint Video Team, Draft ITU-T Recommendation and fianl Draft International Standard of Joint Video Specification, ITU-T Rec.H.264 and ISO/IEC 14496-10 AVC, May, 2003.

[2] C. C. Lin, J. W. Chen, H. C. Chang, Y. C. Yang, Y. H. O. Yang, M. C. Tsai, J. I. Guo, and J. S. Wang, "A 160K Gates/4.5 KB SRAM H.264 Video Decoder for HDTV Applications," IEEE Journal on Solid-state Circuits, vol.42, no.1, Jan. 2007, pp.170-182.

[3] R. Wang, M. Li, J. Li, and Y. Zhang, "High Throughput and Low Memory Access Sub-pixel Interpolation Architecture for H.264/AVC HDTV Decoder," IEEE Trans. Consumer Electronics, vol.51, no.3, Aug. 2005, pp. 1006-1013.

[4] K. Xu and C. Choy, "A Power-Efficient and Self-Adaptive Prediction Engine for H.264/AVC Decoding", IEEE Trans. VLSI, vol.16, no. 3, MARCH, 2008, pp. 302-313.

[5] Y. LEI, H. LI, Z. Zheng etc, "A H.264 Video Decoder with Scheme of Efficient Bandwidth Optimization for Motion compensation", ISCIT. 2007，Oct.17-21, pp. 531-534.

[6] J. H. Kim, G. H. Hyun, and H. J. Lee, "Cache organization for H.264/AVC Motion Compensation", RTCSA.2007, Aug. 2007, pp. 534-541.

[7] T. Chuang, L. Chang, Tsai-Wei Chiu, Y. Chen, and L. Chen, "Bandwidth-efficient cache-based motion compensation architecture with DRAM-friendly data access control", IEEE, ICASSP. 2009, pp.2009-2012.

[8] X. Chen, P. Liu, J. Zhu, D. Zhou and S. Goto, "Block-pipelining cache for motion compensation in high definition H.264/AVC video decoder", IEEE International Symposium on Volume, Issue, 24-27 May 2009 pp.1069 – 1072.

[9] K. Yang, C. Zhang, G. Du, J. Xie, Z. Wang, "A Hardware-Software Co-design for H.264/AVC Decoder", IEEE Asia Solid-State Circuit Conf. Nov. 2006. pp.119-122.

[10] Daniel. F. Finchelstein, Vivienne Sze, Anantha P. Chandrakasa, "Multicore processing and efficient on-chip caching for H.264 and future video decoders", IEEE transactions on circuits and systems for video technology, vol.19, no.11, Nov. 2009, pp. 1704-1722.