A System Architecture Exploration on the Configurable HW/SW Co-design for H.264 Video Decoder

Guo-An Jian, *Jui-Chin Chu*, *Ting-Yu Huang*, *Tao-Cheng Chang*, and *Jiun-In Guo*

Department of Computer Science and Information Engineering, National Chung Cheng University, Chia-Yi, Taiwan, R.O.C.

E-mail: {chienka, cjc, htyu95m, u93410075, jiguo}@cs.ccu.edu.tw

Abstract - In this paper we focus on the design methodology to propose a design that is more flexible than ASIC solution and more efficient than the processor-based solution for H.264 video decoder. We explore the memory access bandwidth requirement and different software/hardware partitions so as to propose a configurable architecture adopting a DEM (Data Exchange Mechanism) controller to fit the best tradeoff between performance and cost when realizing H.264 video decoder for different applications. The proposed architecture can achieve more than three times acceleration in performance.

I. INTRODUCTION

With the progress of science and technology, there are more and more multimedia applications realized on embedded systems. Multimedia applications typically involve the transfer of large amounts of data. Therefore, compression of video, audio, and image data is essential for a cost-efficient use of existing communication channels and storage media. The progress of video compression techniques is getting mature after years of effort on developing many standards. The well-known Moving Picture Expert Group (MPEG) and the Video Coding Experts Group (VCEG) have worked out the latest version of video codec, H.264 [1], which consumes relatively low bit-rate but yields high quality, as compared with the widely popular video CODEC of MPEG series.

In previous works, most researches focused on specific applications such as HDTV, Mobile TV, and so on. They have concentrated on how to meet video compression and processing requirements since video processing require significant amount of computation power. So they proposed solutions, either Application Specific Integrated Circuit (ASIC) [2-3] or processor-based [4-11] solutions, to achieve this target. The ASIC solution offers the best speed performance. But it is limited by its inflexible hardware structure for various requirements on applications. In addition to the ASIC solution, the processor-based solution is the other feasible solution. Due to the limitations of ASIC solutions and the growing interests in computationally intensive multimedia applications, many general-purpose processors for embedded systems now have multimedia extensions. However, the processors adopted in embedded systems are always lack of computational power and cannot achieve the real-time processing requirement.

For this reason, we focus on the exploration of memory access requirement for different software/hardware partitions in an application-specific processor architecture. We propose a configurable architecture adopting the DEM (Data Exchange Mechanism) controller. Such a configurable architecture is more flexible than the ASIC solution and more efficient than the processor-based solution as well. Moreover, users can consider their requirements and modify the system parameters to make the best tradeoff between performance and cost.

The rest of this paper is organized as follows. In Section II, we propose a configurable architecture adopting the DEM controller. Then we perform the design exploration of memory access and analyze different software/hardware partitions on the proposed design in Section III and IV, respectively. In Section V, we evaluate the performance of the proposed design. Finally, we conclude this paper in Section VI.

II. PROPOSED CONFIGURABLE ARCHITECTURE

In the proposed configurable architecture, DEM controller is the most important component that serves as the bridge between the software (tasks executed in processors) and the hardware accelerators. As shown in Fig. 1, DEM controller is the only one master in the architecture and the other hardware accelerators are all slaves. Hence, DEM controller dominates all the I/O access of the hardware accelerators. On the other hand, DEM controller will also dispatch the data and the parameters passed by the processor to the corresponding hardware accelerators. As a result, users can add or delete hardware accelerators easily since there is no data dependency among hardware accelerators. H.264 video decoding can be partitioned into several stages. As shown in Fig. 2, parallel processing will be carried out because each stage of the macroblock can execute its computation concurrently with the stages of other macroblocks. In consequence, the proposed architecture adopting the DEM controller will provide both the flexibility and the computation power.

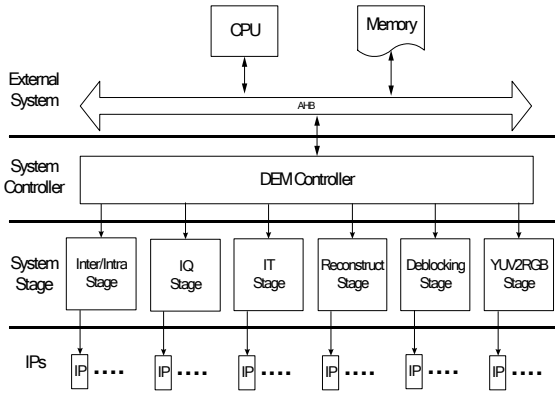

Fig. 1. Proposed configurable architecture for H.264 video decoder

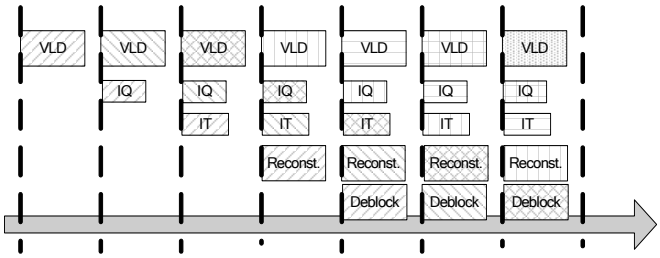

Fig. 2. An example for the processing schedule in the proposed design

III. EXPLORATION ON MEMORY ACCESS

Memory access is an important issue in HW/SW co-design for H.264 video decoding. Major memory access often falls on loading reference data. In this section, we will discuss several methods about minimizing memory access for the bandwidth bottlenecks,

such as intra prediction, inter prediction, and deblocking.

A. Intra Prediction

In H.264 intra prediction, luminance data will be processed in Intra4×4 type or Intra16×16 type. However, chrominance data will be processed only in Intra8×8 type since it is usually smoother than luminance data.

In order to realize a software and hardware partition system, we should consider about where the reference data come from. In Intra4 ×4 prediction, modes 0, 2, 3, 4, 5, 6, 7 need the reference data from the upper macroblock and modes 1, 2, 4, 5, 6, 7, 8 need the reference data from the left macroblock. In Intra16×16 modes, mode 0 uses the reference data from the left macroblock and mode 1 needs the data from the upper macroblock, but mode 2 and 3 need both. Therefore, we explore three solutions for getting the reference data.

In the first solution, we get all the reference data from the external memory as shown in Fig. 3(a). This solution does not need any extra local memory. All the reference data will be grabbed from outside.

In the second solution, we get all the reference data from the internal memory as shown in Fig. 3(b). In this way, we need a buffer whose size is one row of a frame to store the upper reference data and a 16-byte buffer to store the reference data of the left macroblock. The advantage of this solution is the reduction of the external memory access because all the reference data have been stored in the local buffer. But such a solution also has two shortcomings. First, the buffer is very huge when decoding high-resolution video. Second, the buffer is usually idle during decoding P frames.

In the third solution, we get the reference data of the left macroblock from the internal memory and get the upper reference data from the external memory as shown in Fig. 3(c). In this way, it just needs a 16-byte extra local buffer to store the rightmost column data of the left macroblock. Once the prediction mode does not need the reference data from the upper macroblock, we can omit accessing external memory.

We analyzed these three solutions by using SystemC coded hardware modules and C code to do hardware/software co-simulation. Table 1 shows the analysis result. We can see that solution 3 is a better method among them under the consideration between cost and performance.

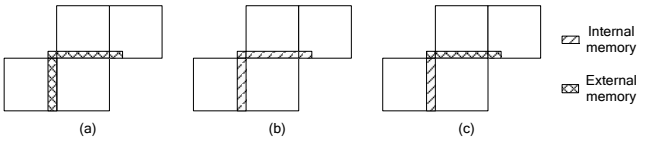

Fig. 3. The way of getting the reference data under different solutions

Table 1. Memory access analysis of intra prediction

	Extra internal memory	Required access data (32 bit bus)	Data bandwidth reduction
Sol. 1	0 byte	10 words	-
Sol. 2	(ImageWidth+16) bytes	0 words	100%
Sol. 3	16 bytes	6 words	40%

B. Inter Prediction

The realization of motion compensation on data blocks with 1/4-pel motion vectors in H.264 reference software (JM) [12] fetches reference data in unit of 9×9 pixels for each 4×4 block without eliminating the redundant data access among neighboring blocks. Fig. 4 shows the data fetching of 4×4 block. As shown in Fig. 5, redundant reference data will be fetched since the data fetching for any type of block is always based on the data fetching of 4×4 block. However, according to our experience, motion compensation using large block size occupies higher probability than that using small block size. In consequence, a large amount of data will be redundantly fetched.

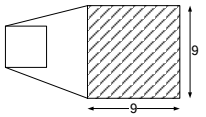

Fig. 4. The data fetching of the 4×4 block

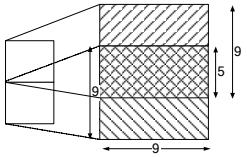

Fig. 5. Overlapped reference data caused by the data fetching of the 4×8 block

Therefore, we have another solution for reducing the data bandwidth for inter prediction, i.e. Variable Block Size Motion Compensation (VBSMC). The VBSMC scheme is proposed to provide the flexibility to respectively fetch reference data in units of 21×21, 21×13, 13×21, 13×13, 13×9, 9×13, and 9×9 pixels for data encoded by 16×16, 16×8, 8×16, 8×8, 8×4, 4×8, and 4×4 types.

As shown in Table 2, comparing the proposed VBSMC scheme to the design adopting only 4×4 block size in the memory access, there are 27%, 33%, 52% , 62%, 64% and 70% reduction of the memory access cycles in the 4×8, 8×4, 8×8 , 8×16, 16×8 and 16×16 modes, respectively. In summary, VBSMC scheme can eliminate the overlapped access and contribute to 48% reduction of data bandwidth.

C. Deblocking

In H.264 decoder, the deblocking filter is applied to each decoded macroblock for the sake of reducing block effects. It smoothes block edges so as to improve the appearance of the decoded frames. Fig. 6 shows the edges filtered by the deblocking filter in a macroblock.

Table 2. Memory access analysis of VBSMC scheme

	Required access data (32 bit bus)	Data bandwidth reduction
4×4 block only	3×9×16 blocks	-
Merge to 4×8 block	3×13×8 blocks	27%
Merge to 8×4 block	4×9×8 blocks	33%
Merge to 8×8 block	4×13×4 blocks	52%
Merge to 8×16 block	4×21×2 blocks	62%
Merge to 16×8 block	6×13×2 blocks	64%
Merge to 16×16 block	6×21×1 blocks	70%

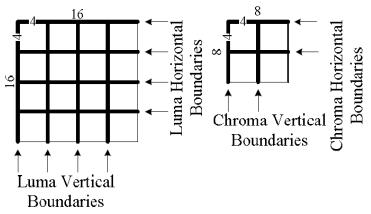

Fig. 6. The edges filtered by the deblocking filter in a macroblock

When filtering the top of horizontal edges, the last four row data of the upper macroblock will be filtered. On the other hand, the rightmost four column data in the left macroblock will be filtered when filtering the leftmost of vertical edges. For this reason, we should consider about where the data come from. As a result, we explore three solutions for getting the reference data.

In the first solution, we get all the reference data from the external memory as shown in Fig. 7(a). This solution does not need any extra local memory. All the reference data will be grabbed from outside.

In the second solution, we get all the reference data from the internal memory as shown in Fig. 7(b). In this way, we need a buffer whose size is four rows of a frame to store the reference data of the upper macroblock and a 64-byte buffer to store the reference data of the left macroblock. The advantage of this solution is the reduction of the external memory access because all the reference data have been stored in the local buffer. But its shortcoming is the buffer will be very huge when decoding high-resolution video.

In the third solution, we get the reference data of the left macroblock from the internal memory and get the reference data of the upper macroblock from the external memory as shown in Fig. 7(c). In this way, it just needs a 64-byte extra local buffer to store the rightmost column data of the left macroblock.

We also analyzed these three solutions by using SystemC coded hardware module and C code to do hardware/software co-simulation. The analysis result is shown in Table 3. We can see that solution 3 is the best choice among them. This is similar to the analysis result of intra prediction.

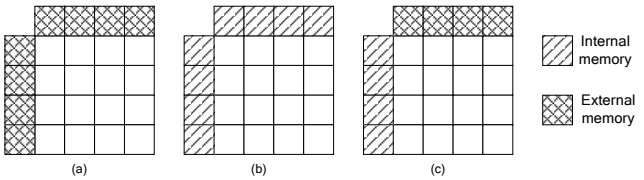

Fig. 7. The way of getting the reference blocks under different solutions

Table 3. Memory access analysis of deblocking

	Extra internal memory	Required access data (32 bit bus)	Data bandwidth reduction
Sol. 1	0 byte	32 words	-
Sol. 2	(ImageWidthx4+64) bytes	0 words	100%
Sol. 3	64 bytes	16 words	50%

IV. Exploration on Software and Hardware Partitions

In this section, we propose several configurations of software and hardware partition for H.264 decoder and analyze the data exchange between hardware and software.

A. Partition 1

According to the complexity profiling of H.264 decoder, we refer the intra prediction and inter prediction to the hardware part since they occupy the most part of computation. Based on this partition, we modified the reference software and created hardware model using SystemC. The system architecture is shown in Fig. 8. The architectures of the following three partitions are similar to the one of partition 1 but have some differences in software and hardware levels. As shown in Table 4, we make a summary to show the implementation details of each function under different partitions.

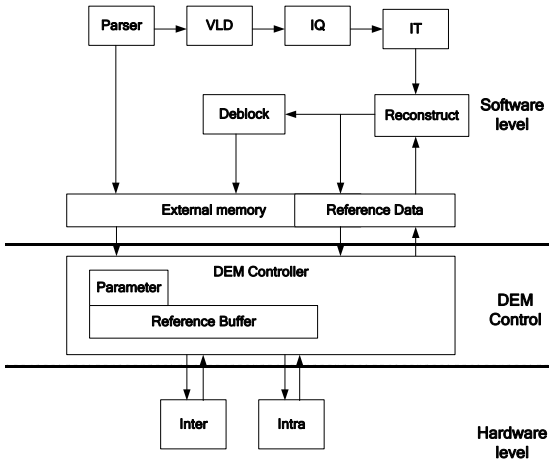

Fig. 8. Architecture of partition 1

Table 4. The implementation of each function under different partitions

	VLD	IQ	IT	Intra	Inter	Reconst.	DB
Partition 1	SW	SW	SW	HW	HW	SW	SW
Partition 2	SW	SW	SW	HW	HW	HW	SW
Partition 3	SW	HW	HW	HW	HW	HW	SW
Partition 4	SW	HW	HW	HW	HW	HW	HW

B. Partition 2

Based on partition 1, we add the second most part of computation, i.e. deblocking, into the hardware part in partition 2. The most important issue of partition 2 is the data exchange between the external memory and the internal memory. Reconstruct IP needs the output data of IT module from the external memory and Deblock IP needs the previous reconstructed macroblock from the internal memory and the upper macroblock from the external memory. Inter IP needs to fetch reference data from the external memory pointed by the motion vector and Intra IP needs the previous macroblock from the internal memory and the upper macroblock from external memory.

C. Partition 3

In partition 3, we consider about the memory access of Deblock IP so that we move Deblocking to the software part and added IQ and IT into the hardware part. For Reconstruct IP, two input data, i.e. reference data and residual data, both come from the internal memory. Reconstructed data will just be exported to the external memory so that Deblocking module can get them. The major memory access rises when IQ fetches the output of VLD,

Inter/Intra fetches the reference data from the external memory, and Reconstruct IP exports the reconstructed data to the external memory.

D. Partition 4

In partition 4, we add most components into the hardware part, except VLD. In this partition, the major memory rises when IQ fetches the output data from VLD, Intra/Inter fetches reference data, and Deblock fetches the reference data from the upper macroblock. This partition has the most hardware accelerators and the maximal size of the internal memory. It also has the best performance and less memory access of system.

V. Verification and Performance Evaluation

In this section we discuss the simulation environment and experimental results by adopting different partitions and configurations. Here we use ARM926EJS virtual platform on SoC Designer to evaluate performance of the proposed partitions and configurations. The simulator is able to report CPU core cycles, total instructions and memory access. The CPU target frequency is set to be 200 MHz.

Fig. 9 shows the co-design verification flow. In step 1, we make a decision for the hardware components and then we construct the hardware model by SystemC in step 2. In step 3, we modify the software structure to fit the hardware part. In step 4, we perform firmware coding so as to exchange information between the hardware and the software part. After finishing the steps mentioned above, we perform the simulation that runs software and hardware co-design system on SoC Designer. Finally, we check the execution results and make the profiling.

According to Section IV, we have provided four partitions for realizing H.264 video decoder. We implement them and put them on the virtual platform of SoC Designer for evaluating the performance. The test video sequences are Foreman and Akiyo and the bit-rate is set 128kbps and 256kbps for QCIF resolution and set 256kbps and 512kbps for CIF resolution. The results of software and hardware co-simulation under different system architectures are shown in Fig. 10. Fig. 11 shows the average performance. Under partition 4, we can get more than three times acceleration in performance.

VI. Conclusion

In this paper, we explore memory access bandwidth and different software/hardware partitions for H.264 video decoder and propose a configurable architecture adopting the DEM (Data Exchange Mechanism) controller. The proposed architecture can achieve more than three times acceleration in performance. Such a configurable architecture involves the advantages of both ASIC and processor-based solutions so that users can make the best trade-off between performance and cost when realizing any application of H.264 video decoder.

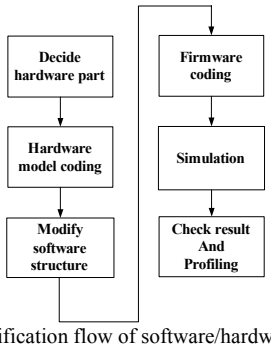

Fig. 9. Verification flow of software/hardware co-design

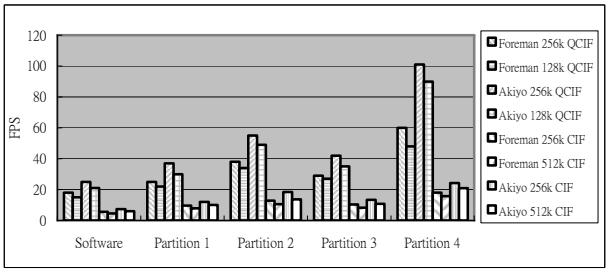

Fig. 10. Performance of different kinds of video sequences under different system architectures

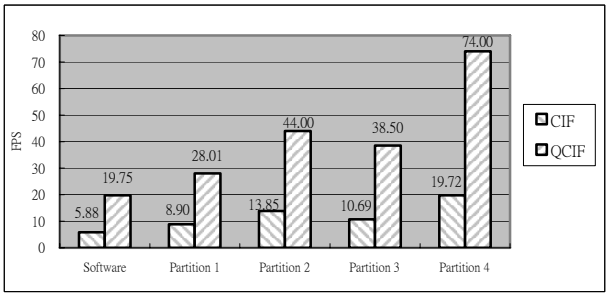

Fig. 11. Average performance under different system architectures

References

[1] ITU-T Recommendation H.264 & ISO/IEC 14496-10, "Advanced Video Coding for Generic Audiovisual Services", Version 4, 2005.

[2] Y. Kun, Z. Chun, D. Guoze, X. Jiangxiang, and W. Zhihua, "A Hardware-Software Co-design for H.264/AVC Decoder", *IEEE Asian Solid-State Circuits Conference*, pp. 119-122, Nov. 2006.

[3] Y. Hu, A. Simpson, K. McAdoo, and J. Cush, "A high definition H.264/AVC hardware video decoder core for multimedia SoC's", *IEEE International Symposium on Consumer Electronics*, pp. 385-389, Sep. 2004.

[4] Q. Xue, J. Liu, S. Wang, and J. Zhao, "H.264/AVC baseline profile decoder optimization on independent platform", *International Conference on Wireless Communications, Networking and Mobile Computing*, vol. 2, pp. 1253-1256, Sep. 2005.

[5] Z. Wei, K. L. Tang, and K. N. Ngan, "Implementation of H.264 on Mobile Device", *IEEE Transactions on Consumer Electronics*, vol. 53, no. 3, pp. 1109-1116, Aug. 2007.

[6] G. Berger, R. Goedeken, and J. Richardson, "Motivation and Implementation of a Software H.264 Real-Time CIF Encoder for Mobile TV Broadcast Applications", *IEEE Transactions on Broadcasting*, vol. 53, no. 2, pp. 584-587, Jun. 2007.

[7] L. Zhuo, Q. Wang, D. D. Feng, and L. Shen, "Optimization and Implementation of H.264 Encoder on DSP Platform", *IEEE International Conference on Multimedia and Expo*, pp. 232-235, Jul. 2007.

[8] C. Peng, H. Wang, C. Li, and Q. Zhang, "The Optimization of H.264 Encoder Based On TI TMS320DM642", *International Conference on Future Generation Communication and Networking*, vol. 1, pp. 38-42, Dec. 2007.

[9] F. Pescador, M. J. Garrido, C. Sanz, E. Juarez, M. C. Rodriguez, and D. Samper, "A real-time H.264 MP decoder based on a DM642 DSP", *IEEE International Conference on Electronics, Circuits and Systems,* pp. 1248-1251, Dec. 2007.

[10] V. Ramadurai, S. Jinturkar, M. Moudgill, and J. Glossner, "Implementation of H.264 decoder on Sandblaster DSP", *IEEE International Conference on Multimedia and Expo*, pp. 694-698, Jul. 2005.

[11] H. C. Lin, Y. J. Wang, K. T. Cheng, S. Y. Yeh, W. N. Chen, C. Y. Tsai, T. S. Chang, and H. M. Hang, "Algorithms and DSP implementation of H.264/AVC", *Asia and South Pacific Conference on Design Automation*, pp. 742-749, Jan. 2006.

[12] Available via http://iphome.hhi.de/suehring/tml/