A HW/SW Co-design Methodology for Video Compression Algorithms

Sungjei Kim[1], Byoungho Kim[2], Jaehwan Joo[1], Yungho Choi[2], and Yoonsik Choe[1]
[1] Electrical and electronic engineering department of Yonsei University, Seoul, Korea.
e-mail: {coldeyes, jhjoo80, and yschoe}@yonsei.ac.kr
[2] Electrical engineering department of Konkuk University, Seoul, Korea.
e-mail: tizimah@naver.com, yunghoch@konkuk.ac.kr

Abstract- **Many hardware and software co-design methodologies have been proposed to improve overall system performance, reliability and cost-effectiveness of digital systems for decades. This paper presented an efficient co-design methodology using primitive instruction sets and a dynamic out-of-order execution scheduler for a variety of video compression algorithms. As a case study, this paper applies the proposed co-design method to the motion estimation of H.264 video coding standard, which is one of the most time-consuming parts in H.264. This case study shows that the proposed methodology improves motion estimation speed by 13 times compared to naïve motion estimation with a minimal design and coding effort.**

I. INTRODUCTION

Recently, demands of various multimedia services have been radically increased. Such service demands require efficient video compression algorithms to efficiently and economically store and transmit multimedia contents. For this, many video compression standards such as MPEG-2 [1], MPEG-4 [2], and H.264 [3] have been proposed and used, during last decades. However, since the high compression efficiency of these standards is mainly due to high computation complexity, it is very difficult to implement them in software for real-time applications. To resolve this problem and thus, to provide real-time video compression encoding with a little quality degradation, full hardware design solutions have been proposed [4], [5], [6]. However, these dedicated hardware architectures have a problem of short flexibility and long time-to-market.

To overcome the problems described above, many hardware and software co-design solutions have been proposed, which can take advantages of both software and hardware solutions, i.e., flexibility and performance [7], [8], [9]. Jang *et al*. [7] presented a H.263 video compression codec [10] which implements motion estimation and motion compensation parts of H.263 codec in hardware and processes the rest of H.263 in software. Choi *et al*. [8] also presents a MPEG-4 hardware and software co-design solution which proposes to design a codec in hardware except for a variable length coding (VLC) part of MPEG4. Even though the quality and performance of the proposed algorithms given above are tolerable and acceptable, their bulky-sized hardware IP blocks such as ME or MC parts, degrade the benefits of software solutions, i.e., flexibility and low development cost.

In order to provide the design flexibility of previous works, a co-design approach based on the UltraSONIC reconfigurable platform is recently proposed [9], [11]. In [9], Wiangtong *et al*. proposes an algorithm which automatically partitions and schedules tasks for hardware and software, respectively. Because run-time reconfigurable processing elements (PEs) of UltraSONIC can contain any specific data-dominated tasks, various combinations of PEs provide flexibility and reusability to a hardware task design. However, a sub-optimized partitioning algorithm by automated tools degrades the performance of co-design system compared to hand-made. In addition, task manager algorithm which performs scheduling in task unit makes the system limited in a parallel processing of instruction unit and thus the performance degradation is occurred.

In this paper, we propose a hardware and software co-design methodology for various video compression algorithms. The proposed co-design methodology employs primitive instruction sets and an out-of-order execution scheduler and thus, can exploit the benefits of both software and hardware solutions, i.e., flexibility and performance.

In the remainder of this paper, we describe the proposed co-design methodology that consists of primitive instruction sets and out-of-order execution scheme in Section II. In Section III, a case study for the motion estimation part of H.264 will be considered. Finally, we conclude this work while giving future work directions in Section IV.

II. HW/SW CO-DESIGN METHODOLOGY

In this section, we present an efficient co-design method for video compression algorithms. This method takes care of two co-design issues. The first issue is how tasks are partitioned for hardware and software. The other is how to maximize system performance. To handle the first issue, this work introduces a primitive instruction set which enables parallel processing in instruction unit and thus, defines an implementation boundary between hardware and software. Regarding the second issue, this paper employs an out-of-order scheduler for vector instructions or primitive instructions. Before these issues are covered in depth, the following section will cover the overview of the proposed co-design flow.

This research was supported by the Seoul R&BD Program (10654), Korea.

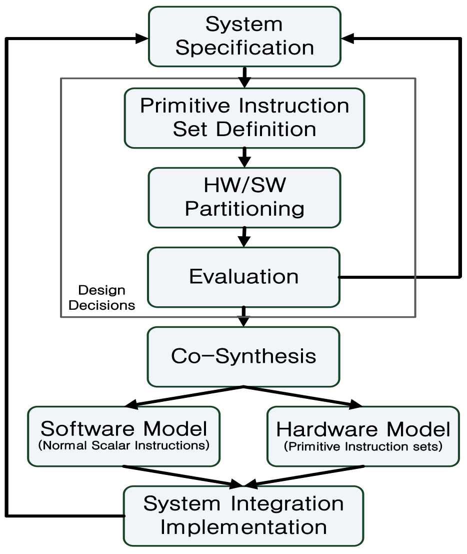
A. *The Proposed HW/SW Co-design Flow*

The proposed co-design flow iteratively partitions hardware and software tasks by defining a primitive instruction set until system requirements are satisfied. For detail, as shown in Fig. 1, a primitive instruction set is determined through analyzing system specifications and application behaviors. Primitive instructions define which tasks should be implemented in hardware and thus, result in efficient HW/SW partitioning. After that, the system design based on the primitive instruction set needs to be evaluated in order to check whether system requirements are satisfied

Until given target requirements such as performance, chip size, cost and power are satisfied, co-design system is iteratively tuned. Therefore, the proposed co-design methodology ensures a tight integration between software and hardware, ending up with better performance and design efficiency. More details about primitive instructions are given in the following section.

B. *Primitive Instruction design methodology*

The key idea of the proposed co-design methodology is to define primitive instructions which are a sort of small function blocks. Each one of these primitive instructions processes a complicated and time-consuming task while being called frequently. Therefore, if these instructions are implemented in hardware, the overall performance of a target system can be easily enhanced. This methodology also can provide a hardware implementation of small-sized flexible task units and enables to efficiently partition hardware and software tasks.



System Specification
Primitive Instruction Set Definition
HW/SW Partitioning
Evaluation
Design Decisions
Co−Synthesis
Software Model
(Normal Scalar Instructions)
Hardware Model
(Primitive Instruction sets)
System Integration Implementation

Fig. 1. Proposed HW/SW Co-design Flow.

Furthermore, from the view of software engineers, they are just a set of new powerful instructions and thus, enable an easy software development by replacing corresponding function calls by the newly-given primitive instructions.

Fig. 2 shows a primitive instruction determination process. In this figure, A, B, C, ..., F are primitive tasks comprising a video application. By clustering, partitioning or isolating of these tasks, candidate primitive instructions can be determined when one of the following conditions is satisfied.

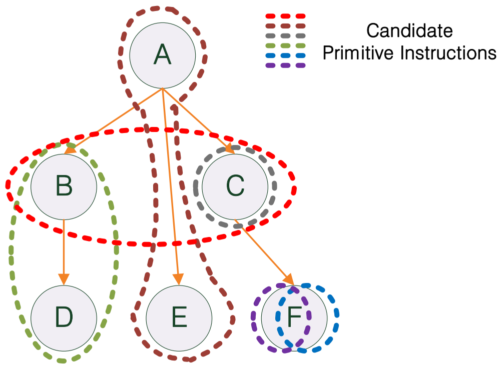


Fig. 2. Candidate Primitive Instruction Decision Procedure.

(1) If an isolated task, like C task in Fig. 2, is an iteratively called and data-intensive function block, the task can be a candidate primitive instruction.
 Examples: SAD or SATD [1] functions in motion estimation.

(2) If a task, like B-C, B-D, or A-E tasks in Fig. 2, can be generated by clustering some frequently called tasks, this new task can be a candidate primitive instruction.
 Examples: A set of scalar instructions, i.e. adders and multipliers.

(3) If a task, like B task in Fig. 2, operates a regular-sized block data access from main memory to register files, and can be implemented in vector processing unit, this task can be a candidate primitive instruction.
 Examples: vector data-load instructions.

But, these candidate primitive instructions are finalized and preferentially selected when one or more of the following conditions are satisfied. If not, candidate primitive instructions are iteratively split or merged until being satisfied.

(1) If the candidate primitive instructions are a highly critical bottleneck to implement real-time video coding systems, they can be finalized as primitive instructions.
 Examples: DCT or interpolation functions.

(2) If the candidate primitive instructions are applicable and flexible to previous video codec, those candidate primitive instructions can be primitive instructions.
 Examples: SAD4x4 unit in SAD function.

---

[1] SAD and SATD are criteria to find a nearest one of current macroblock in the view of distance.

$$SAD = \sum_{i=1}^{N} |x_i - y_i|, \qquad SATD = \sum_{i=1}^{N} |T(x_i - y_i)|$$

, where N is the number of pixels of a macroblock, $T(\cdot)$ is Hadamard transform and x_i, y_i are current and candidate macroblock respectively.

The best or nearest macroblock y_i is chosen when the criterion has a minimum SAD or SATD value.

(3) If the candidate primitive instructions are defined as a possibly dependency minimized set, they can be primitive instructions.

Finally, selected primitive instructions can be implemented in hardware by employing single instruction multiple data (SIMD) or multiple instructions multiple data (MIMD) structure. These structures enable one to process primitive instructions in a short time, resulting in short execution time. Therefore, flexibly defined primitive instructions can provide compatibility to previous codec and higher performance in speed.

To boost up the performance of system based on primitive instructions, the following section will present an out-of-order scheduler for either normal scalar or primitive instructions.

C. *Out-of-Order Execution Scheduler*

With an in-order scheduler, fetched instructions are sequentially executed in a program order pattern. In this case, a stalled instruction blocks the following instructions which do not have to be blocked. For example, assume a "multiplication" function unit which can multiply data in 10 cycle latency. In the case of in-order execution, any following instructions cannot be issued and executed until the multiplication instruction is completed. To resolve such problem, an out-of-order execution scheduler rearranges instruction execution order by dynamically tracking instruction dependencies and checking whether functional units are available. This increases processing resource utilization and reduces execution time. With the example of multiplication instruction given above, in the case of out-of-order execution, if next "add" instruction has no dependency with the prior multiplication instruction, it can be issued and executed without waiting for the multiplication instruction completion, enhancing system performance and utilization.

Generally, such out-of-order scheduler has been employed for scalar instructions rather than vector instructions. This is because it is difficult to track dependencies among vector instructions consisting of many data and variables. However, our primitive instruction is confined to have only one output data, which makes dependency tracking easy. By exploiting this, this work proposes to employ an out-of-order scheduler for both primitive instructions and normal scalar instructions.

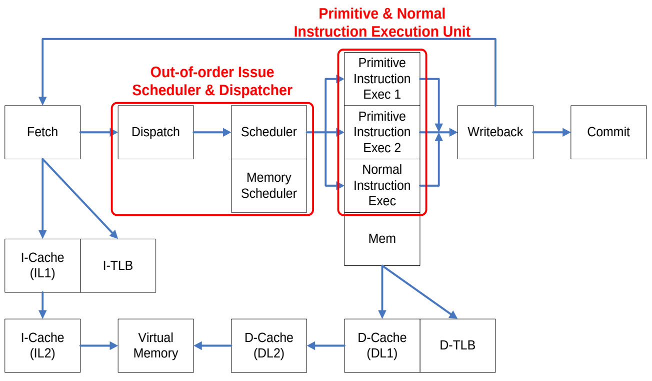


Fig. 3. Out-of-order execution Scheduler.

Fig. 3 shows an architecture example employing primitive instructions and an out-of-order scheduler. In this figure, after fetching instructions from instruction memory, the instructions will be dispatched and sent to reservation station in the out-of-order hardware scheduler. At reservation station, data dependency of instructions and function unit availability are checked and, if there is no problem, instructions are sent to execution units in any order. Then, the results of executed instructions are written back into either registers or buffers.

The proposed out-of-order execution scheduler checks not only data dependency between primitive instructions but also dependency between primitive instructions and normal scalar instructions. Therefore, neither primitive instructions nor scalar instructions block one another, maximizing system utilization and performance.

III. A Case Study: H.264/AVC Motion Estimation

To help understanding, in this chapter, we provide a case study which applies the proposed design methodology to H.264/AVC motion estimation. For this, this chapter analyzes characteristics of H.264/AVC motion estimation and defines primitive instructions for hardware and software partitioning. Then, by exploiting the defined primitive instructions, we design a hardware and software co-design architecture. Finally, the performance of this architecture employing primitive instruction sets and an out-of-order execution scheduler is verified by using “SimpleScalar” simulator.

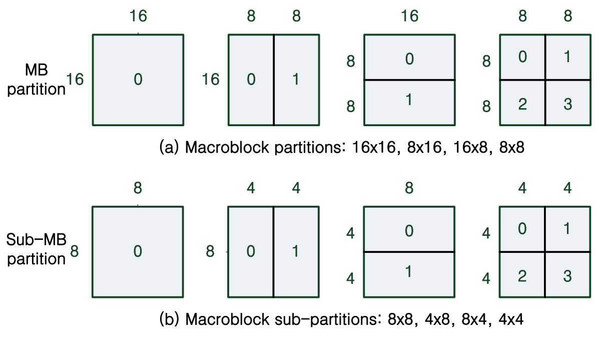

MB partition
16
0
16
8
8
0
1
16
8
8
0
1
8
8
0
1
2
3
(a) Macroblock partitions: 16x16, 8x16, 16x8, 8x8
Sub-MB partition
8
0
8
4
4
0
1
8
4
4
0
1
4
4
0
1
2
3
(b) Macroblock sub-partitions: 8x8, 4x8, 8x4, 4x4

Fig. 4. Macroblock and Sub-macroblock Partitions for Motion Estimation.

A. *Design Specifications of H.264 Motion Estimation*

As a state-of-the-art video coding standard, H.264 provides gains in compression efficiency of up to 50% over a wide range of bit rates and video quality compared to previous standards. However, complexity of H.264 highly increased about 10 times as complex as a corresponding MPEG-4 standard [12]. Specifically, to reduce the temporal redundancy between successive pictures, motion estimation algorithm has improved by adopting variable block size for motion compensation and higher motion vector resolutions.

Compared to previous video coding standards, H.264 has a variable block size motion estimation scheme to represent one macroblock. Fig. 4 shows the candidate macroblock and sub-macroblock partitioning from Inter16x16 to Inter4x4 mode. A macroblock is composed of 16x16 pixels, and it can be divided into two 16x8 partitions, two 8x16 partitions or four 8x8 partitions. If the 8x8 partitions are selected, each of the four 8x8 sub-macroblocks within the macroblock may be split in a further 4 ways like (b) in Fig. 4. As results of iterative partition motion search to find the best mode, the complexity and computation load of motion estimation increased and consumed 60~80% of the total encoding time [13].

To resolve this complexity of motion estimation, this case study applies our proposed co-design methodology to motion estimation unit design. For this, we assume the followings.

(1) 4:2:0 YCbCr format and QCIF (176x144) resolution.
(2) Full search algorithm is used.
(3) Motion vector search range is 16 pixels.
(4) Motion vector resolution is 1/4 pixels per one block.
(5) Matching criteria: SAD for integer-pixel search,
SATD for sub-pixel search.
(6) 5 reference frame number and “Foreman” test sequence.

B. *Primitive Instruction Definition for Motion Estimation*

To define primitive instructions for H.264 motion estimation part as explained in section II, JM 10.1 reference software and Intel Vtune performance analyzer 8.0 are used for profiling [14]. The profiling result of motion estimation is shown in Fig. 5. As shown, H.264 motion estimation consists of four main functions, i.e., SAD for integer-pixel search, SATD for sub-pixel search, interpolation for multiple reference picture generation and miscellaneous instruction part.

Since SAD function is an iteratively called and data-intensive function block, it can be a candidate primitive instruction. But, to support motion estimation scheme in H.264, implementing of all variable size SAD blocks, 16x16, 16x8, 8x16, …, 4x4, are redundant. Moreover, 16x8 size SAD function may not be applicable to motion estimation of MPEG-2 or 4 standards based on 8x8 or 16x16 block motion search. This can make a 4x4 SAD function block as a good primitive instruction candidate because a 4x4 SAD function block makes any kinds of SAD blocks for any video compression standards. Therefore, we define 4x4 SAD function block as SAD4x4 primitive instruction (PI).

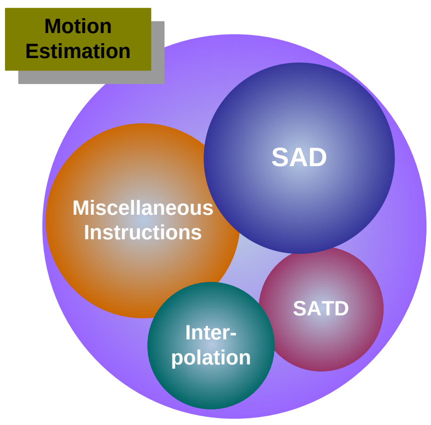


Fig. 5. Profiling Results for Motion Estimation by Intel Vtune Analyzer

SATD is another frequently-called and data-intensive function for sub-pixel motion search, making SATD a primitive instruction candidate. Additionally, this primitive instruction can be used to implement another important H.264 function, i.e., DCT4x4.

Since the data size of the primitive instructions selected above is 4x4 block, data from main memory to primitive instruction registers should be 4x4 vectors. Furthermore, before the SAD4x4 and SATD are calculated, their source data should be loaded into cache as soon as possible. Therefore, we define LoadLine4x4 and LoadLine4x4_2 PIs as primitive instructions for high speed vector data accesses. These LoadLine4x4 and LoadLine4x4_2 PIs can be used to implement a miscellaneous instruction function in Figure 5.

There are many memory address pointers to calculate the image blocks. Since the address of macroblocks or sub-macroblocks in one picture is frequently operated and moved to next on motion estimation process, we can cluster those instructions, which are less dependency with each other, at a distance in software program. These are the NextAdrCalc PIs to calculate the addresses for next memory pointer.

All of PIs mentioned above are listed in Table 1, and some remarks are included for understanding. Interpolation of reference pictures is omitted in this case study. This will be covered in future works.

Table 1. Primitive Instruction sets for ME within H.264

Primitive Instructions	Remarks
SAD4x4	- Minimized SAD unit - Applicable to previous codec - used in integer-pixel search
SATD	- Data-intensive function - Flexible to DCT4x4 - used in sub-pixel search
LoadLine4x4	- Vector memory(VM) access unit - used in integer-pixel search
LoadLine4x4_2	- VM access unit per 4 pixels - used in sub-pixel search
NextAdrCalc	- Set of frequently called instructions at a distance. - for integer-pixel search
NextAdrCalc_2	- Set of frequently called instructions at a distance. - for sub-pixel search

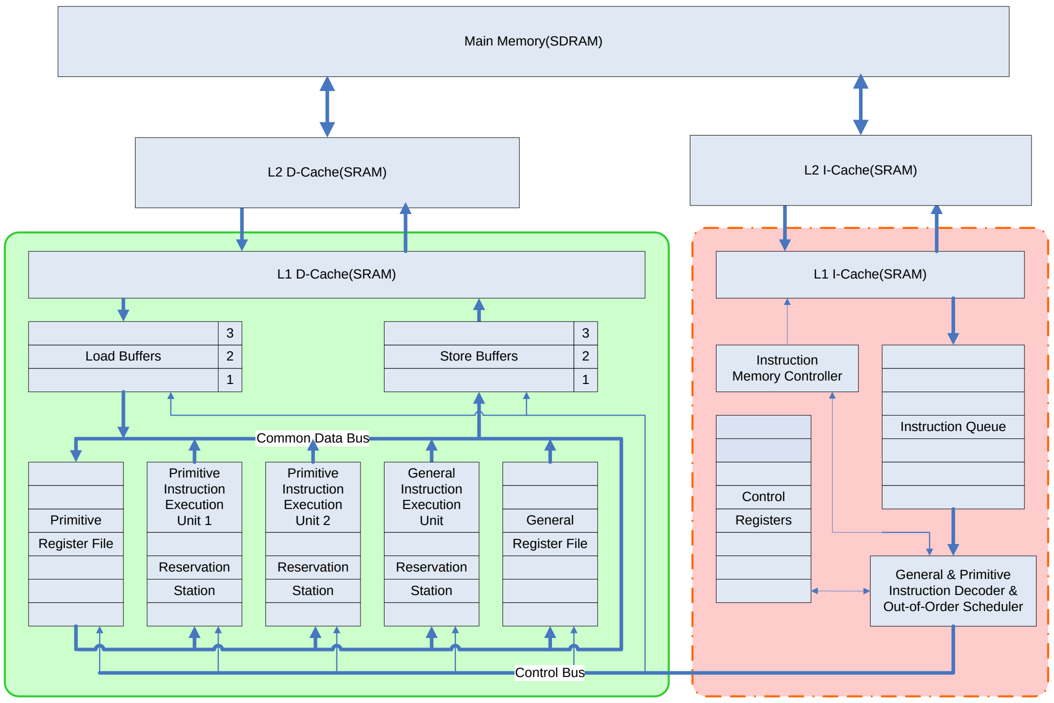


Fig. 6. An implementation of the proposed system architecture.

C. *Implementation of HW/SW Co-design Architecture*

To implement a HW/SW co-design architecture, we choose a general purpose RISC processor, out-of-order scheduler, main memory (SDRAM), L1/L2 Cache (SRAM), common data and control bus, hardware modules, instruction buffers (Queue) and register files to support primitive or normal scalar instructions. All of separated modules mentioned above are mixed and designed in one architecture platform as Fig. 6.

The hardware and software co-design architecture employing primitive instruction sets and out-of-order execution scheduler is shown in Figure 6. This architecture has a RISC (Reduced Instruction Set Computer) structure including an out-of-order scheduler based on Tomasulo's algorithm for scalar instructions [15]. To support an out-of-order execution for primitive instructions, reservation stations and original scheduler are improved.

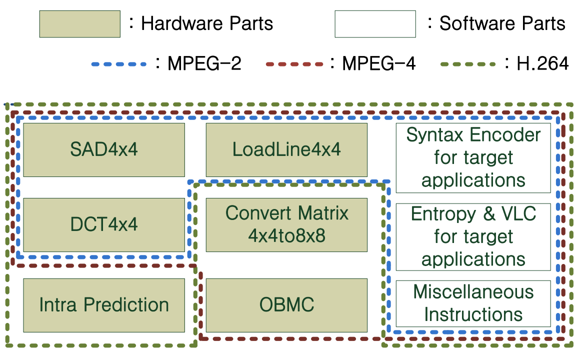
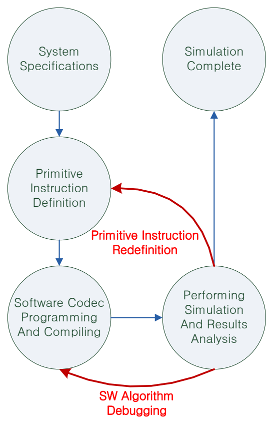


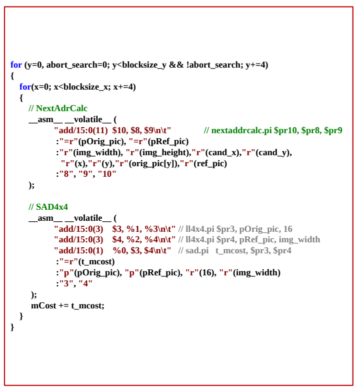
Fig. 7. Combinations of Codec Function Library for video applications.

As shown in Fig. 6, instructions are sent from L1 I-Cache into the instruction queue, which they are issued in FIFO order, by instruction memory controller. The reservation stations include information used for detecting data dependencies for predefined primitive instructions (hardware tasks), as well as general instructions (software tasks). This information enables both primitive instructions and normal scalar instructions to schedule in out-of-order pattern. The load buffers and store buffers make memory data reorders in a program order pattern and read or write. The data and controls needed to load, execute and store are communicated through the common data and control bus.

This target architecture template can be easily extended and customized for a range of video applications allowing the primitive instruction sets to be reformed. Fig. 7 shows the example of how the proposed architecture can be easily adapted to various video applications such as MPEG-2, MPEG-4 and H.264. The combination of some primitive instruction sets can be a specific video codec. H.264, for example, consists of SAD4x4, LoadLine4x4, DCT4x4 and Intra Prediction implemented and executed in hardware parts and Syntax Encoder, Entropy Coding, VLC and other instructions processed in software. Since the out-of-order execution scheduler accelerates a system, by selectively choosing the primitive instruction sets from codec function library like in Fig. 7, software engineers do not need to fully understand a specific hardware architecture using MMX or VLIW technology to optimize the software implementation for a target specification.


System Specifications
Simulation Complete
Primitive Instruction Definition
Primitive Instruction Redefinition
Software Codec Programming And Compiling
Performing Simulation And Results Analysis
SW Algorithm Debugging

(a)



```
for (y=0, abort_search=0; y<blocksize_y && !abort_search; y+=4)
{
  for(x=0; x<blocksize_x; x+=4)
  {
    // NextAdrCalc
    __asm__ __volatile__ (
          "add/15:0(11)  $10, $8, $9\n\t"          // nextaddrcalc.pi $pr10, $pr8, $pr9
          :"=r"(pOrig_pic), "=r"(pRef_pic)
          :"r"(img_width), "r"(img_height),"r"(cand_x),"r"(cand_y),
            "r"(x),"r"(y),"r"(orig_pic[y]),"r"(ref_pic)
          :"8", "9", "10"
    );

    // SAD4x4
    __asm__ __volatile__ (
          "add/15:0(3)   $3, %1, %3\n\t" // ll4x4.pi $pr3, pOrig_pic, 16
          "add/15:0(3)   $4, %2, %4\n\t" // ll4x4.pi $pr4, pRef_pic, img_width
          "add/15:0(1)   %0, $3, $4\n\t"   // sad.pi   t_mcost, $pr3, $pr4
          :"=r"(t_mcost)
          :"p"(pOrig_pic), "p"(pRef_pic), "r"(16), "r"(img_width)
          :"3", "4"
     );
     mCost += t_mcost;
  }
}
```

(b)

Fig. 8. (a) Simulation and Analysis Procedure in SimpleScalar architecture. (b) Primitive Instruction Example in one part of H.264 ME.

D. *Simulation Results*

To verify and evaluate performance of the proposed hardware and software co-design methodology, that was modeled by using SimpleScalar simulator, which is developed and supported by T. Austin at SimpleScalar LLC [16].

SimpleScalar is one of the execution-driven simulators and supports out-of-order execution and user-extensible instruction format [17]. Therefore, our proposed PIs and out-of-order execution scheduler are tested well in this simulator. Fig. 8 (a) shows the procedure of simulation and analysis with proposed primitive instructions in the SimpleScalar simulator. Fig. 8 (b) shows a specific example of primitive instructions, NextAdrCalc and SAD4x4, written in inline assembly codes.

To verify performance of proposed co-design architecture, we set following 3 test conditions and performed them in SimpleScalar simulator. The first condition performs motion estimation of in-order scheduling without primitive instruction sets. The second one performs motion estimation of an in-order execution structure with primitive instruction sets. The last one performs motion estimation of out-of-order scheduling with primitive instruction sets. In simulation process, predefined primitive instruction sets are assumed that execution is processed in one cycle unit by fully allocating the hardware resources.

Simulation results of H.264 motion estimation are as following Table 2. In the original H.264 with in-order issue structure, the total cycles are 22.57 billion cycles, but after partial function blocks of motion estimation are replaced by proposed primitive instructions, the performance is 4.93 times increased within in-order execution. The cause of this improvement is that the primitive instructions enable to process the complicated and time-consuming tasks in short time.

In primitive instruction sets and out-of-order structure case, the total cycle is reduced to 1.7 billion cycles. This condition is approximately 13 times faster than naive motion estimation algorithm, and 2.62 times faster than primitive instruction sets with in-order execution. Because the out-of-order scheduler boosts up the performance of system based on primitive instructions, the amount of resource usages in each pipe stage have increased and this result caused the reduction of total cycles.

Instruction Per Cycle (IPC) in Table 2, is a measure to know how the amount of parallelism among instructions exists. The higher IPC value means that the number of instructions performed per one cycle is higher. Therefore, the parallel processing of instructions in this case is performed well. IPC results with in-order execution scheduler in Table 2 show the less difference between original and primitive instruction sets, but the result with out-of-order execution scheduling shows that the IPC is largely reduced and the parallel processing is performed well.

Table 2. Simulation Results of H.264 ME part.

	Total Cycles	IPC
Original + In-order execution.	22565476649	0.8070
PI sets + In-order execution.	4579174158	0.8444
PI sets + Out-of-order exec.	1747285199	2.2129

IV. Conclusion And Future Work

In this paper, the co-design methodology based on primitive instruction sets and out-of-order execution scheme is proposed, which can take the advantages of both software and hardware; flexibility, low development cost, compactness and adequately high performance.

By properly defining of primitive instruction sets, the proposed methodology can provide the flexibility to previous video coding standards and low cost to system development. Since the primitive instruction sets implemented in hardware units are reordering with general instructions by out-of-order execution scheduler, the proposed co-design method sufficiently accelerates any video coding algorithms in the encoding speed view.

Moreover, since the primitive instruction sets can support a powerful codec function library, software engineers do not need to fully understand the specific hardware architecture using MMX or VLIW technology to optimize the software implementation for a target application.

In a case study for H.264 motion estimation part, our proposed co-design method has increased the effect of time saving more than 13 times to the original motion estimation system. This result can be more improved by how efficiently defining of primitive instruction sets. In the light of view, the generalization of primitive instruction sets is still the main subject of our ongoing study.

In a future work, we will apply the proposed design methodology to other parts of H.264 video codec standard such as reference picture interpolation, intra prediction, rate-distortion optimization mode decision. In addition, a prototype system of this easily extended and customized co-design methodology for a range of video coding standards will be implemented in FPGA and verified.

REFERENCES

[1] ISO/IEC 13818-2: "Information technology – Generic coding of moving pictures and associated audio information: video," 1996.

[2] ISO/IEC 14496-2: "Information technology – Coding of audio-visual objects- part2: Visual," 1999.

[3] ISO/IEC 14496-10: "Coding of Audiovisual Objects-Part 10: Advanced Video Coding," Dec. 2003.

[4] M. Irfan, A. K. Khan, and H. Jamal, "FPGA based implementation of MPEG-2 compression algorithm," *IEEE 17th Int. Conf. On Microelectronics*, pp 204-244, Dec. 2005.

[5] K. Denolf, C. D. Vleeschouwer, R. Turney, G. Lafruit, and J. Bormans. "Memory centric design of an MPEG-4 Video Encoder," *IEEE Trans. On Circuits and Systems for Video Technology*, Vol. 15, No. 5, May 2005.

[6] T. Chen, S. Chien, T. Huang, C. Tsai, C. Chen. T. Chen, L. Chen, "Analysis and architecture design of an HDTV720p 30 frames/s H.264/AVC encoder," *IEEE Trans. On Circuits and Systems for Video Technology*, Vol. 16, No. 6, Jun. 2006.

[7] S. K. Jang, S. D. Kim, J. Lee, G. Y. Choi and J. B. Ra, "Hardware-software co-implementation of a H.263 video codec," *IEEE Trans. on Consumer Electronics*, Vol. 46, pp.191-200, Feb. 2000.

[8] J. Choi, N. Togawa, T. Ikenaga, S. Goto, M. Yanagisawa and T. Ohtsuki, "An efficient algorithm/architecture codesign for image encoders," *IEEE 47th Int. Midwest Symp. On Circuits and Systems*. Vol. 2, pp. 469-472, Jul. 2004.

[9] T. Wiangtong, P. Y. K. Cheung, and Wayne Luk, "Hardware/Software Codesign – A systematic approach targeting data-intensive applications," *IEEE Signal Processing Magazine*, Vol. 22, No. 3, May, 2005.

[10] Draft ITU-T Recommendation H.263, "Video coding for low bit-rate communication," Mar. 1996.

[11] S. D. Haynes, H. G. Epsom, R. J. Cooper, and P. L. McAlpine, "UltraSONIC: A reconfigurable architecture for video image processing," in *Proc. Field-Programmable Logic and Applications*, pp. 482-491, 2002.

[12] T. Wiegand, G. J. Sullivan, G. Bjontegaard and A. Lutha. "Overview of the H.264/AVC Video Coding Standard," *IEEE Trans. On Circuits and Systems for Video Technology*, Vol.13, No.7, pp.560-576, Jul. 2003.

[13] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi, "Video coding with H.264/AVC: tools, performance, and complexity," *IEEE Circuits and Systems Magazine*, Vol. 4, pp.7-28, First Quarter, 2004.

[14] Reference Software: available at http://iphome.hhi.de/suehring/tml/

[15] R. M. Tomasulo, "An efficient algorithm for exploiting multiple arithmetic units", IBM Journal of Research and Development, Vol. 11, No. 1, pp. 25`33, Jan. 1967.

[16] SimpleScalar LLC web Site at http://www.simplescalar.com/

[17] T. Austin, E. Larson, and D. Ernst, "Simplescalar: An infrastructure for computer system modeling," *IEEE Computer Society Magazine*, Vol. 35, No. 2, pp. 59-67, Feb. 2002.