A Design Optimization Method for ASIPs including On-Chip Cache Memories

Doctor's Thesis

by

Jun Sato
Department of Electrical Engineering
Tsuruoka National College of Technology

Supervisor:
Professor Masaharu Imai
Reviewers:
Professor Akihiro Hashimoto
Professor Ken-ichi Taniguchi

Contents

Abstract

The integration density of VLSI (Very Large Scale Integration) is steadily increasing every year, and the performance of VLSI is also improving. In the year 2012, feature size (design rule), highest clock frequency, and density on microprocessor chip are predicted as 0.05 μm, 10 GHz, and 1.4 Billion transistors, respectively. Then, an Application Specific Integrated Processor (ASIP), which contains a CPU core and peripheral circuits, will also include large memory units (for instruction and data) on the same chip by using Deep Submicron Technology (DSM).

However, as the size of design increases rapidly, the time and effort to design the ASIP would exceed the ability of designers. As a result, so called "design productivity crisis" will become a serious problem. Furthermore, an ASIP would be not only a large hardware system but also a very complicated one, which is composed of large amount of hardware and software components.

One of the key issues in the ASIP design is the optimization of instruction set architecture and CPU architecture. Because the performance of an ASIP is heavily affected by the choice of these architectures, it is essential to select an optimum instruction set architecture and CPU architecture under given design constraints. Consequently, in order to develop an ASIP with specific instructions, it is necessary to generate a set of software tools to develop application programs, which includes a compiler, an assembler and a debugger. These software tools would require a long term effort to develop manually even if efficient software generation tools were used.

Another important problem is the optimization of architecture including memory units in the ASIP. Because the clock frequency of CPU core is usually higher than the access cycle frequency of on-chip memories and is much higher than that of off-chip memories, fast cache memories are required between CPU core and on-chip memory,

and on-chip and off-chip memories, to fill the frequency gap. Furthermore, embedded systems for specific application domain such as for multimedia processing, require huge memory space, then additional external memory would be indispensable to realize such application systems. However, fast and large memory is very expensive. In order to satisfy these requirements, an optimization method to decide an efficient on-chip memory configuration is required.

In an ideal ASIP development environment, the optimization of ASIP architecture should be performed automatically. While many of high-level synthesis systems concentrate on generating hardware design of ASIPs, most of them do not employ any mathematical methods to optimize system architectures.

In this thesis, a method to optimize the design of ASIPs including on-chip memories is proposed. First, a design framework to optimize the ASIP architecture is introduced. Then, an optimization method of on-chip memory for ASIP is described.

The proposed method was realized as PEAS-I (Practical Environment for ASIP Development - type I), which is a hardware/software codesign system for ASIP development. PEAS-I system can decide an optimum instruction set and its implementation method under the given design constraints taking advantage of analytical results of application programs. Thus, PEAS-I system can reduce the execution time of application programs, for example. Moreover, the performance of ASIP can be improved by considering tradeoffs of the amount of hardware of the CPU core, memory, peripheral circuits, etc.

The efficiency and effectiveness of PEAS-I system were confirmed through several experiments. According to the experimental results, PEAS-I system gives accurate estimation of the chip area and performance of ASIPs before the detailed hardware design is completed. The experimental results also show that PEAS-I system is able to generate both hardware design and a set of application program development tools for a typical size ASIP within several hours. Then, the proposed effectiveness of the design methodology was demonstrated through several design examples.

In the second half of this thesis, a performance optimization method is proposed for hierarchical memory system in ASIPs, which consists of on-chip fast cache memory, a large amount of on-chip ordinary memory, and a huge off-chip memory.

Using the hierarchical on-chip memory system, the performance of an ASIP could be improved up to 50 % compared to that with conventional cache memory system.

The performance optimization method includes hit-ratio prediction, write-back penalty prediction and average memory access time estimation. From the experimental results, it is known that the proposed method can decide an optimal configuration of the on-chip memory much more efficiently than conventional optimization methods based on the iteration of cache simulation. The proposed method can estimate the average memory access cycle very accurately for fully associative caches. Even when the cache memory is non-fully associative, the performance of the on-chip memory configurations obtained by the proposed method was found to degrade only up to 5 % compared to that by the conventional cache simulation method.

Acknowledgments

The author would like to express his deepest gratitude to his supervisor Professor Masaharu Imai, Osaka University, for introducing the author to this research area and guiding this work, for providing all the facilities to carry it out, and for continuous support, help and encouragement.

The author also likes to express his thanks to Professor Akihiro Hashimoto and Professor Ken-ichi Taniguchi for reviewing this thesis, and to Professors and staffs of the Department of Informatics and Mathematical Science, Graduate School of Engineering Science, Osaka University for thier kind help.

The author is extremely thankful to Prof. Yoshinori Takeuchi from Osaka University, Prof. Akichika Shiomi from Shizuoka University and Prof. Akira Kitajima from Osaka University for their continuous support, help and encouragement, and many thanks to all members of PEAS project for their kind assistance, especially, Dr. Alauddin Alomary, Dr. N. N. Binh, Mr. Kouichi Fukuda, Mr. Makoto Ichida, Mr. Takeharu Nakata, Dr. Yoshimichi Honma, Dr. Takafumi Kinoshita, Mr. Kunio Honsawa, Mr. Jota Tomita, Dr. Tesuya Hakata, Dr. Tsutomu Kimura, Prof. Takumi Nakano, Mr. Ryoji Sakurai, Mr. Mineo Shoji, and Dr. Yoshihiro Aoyama, and to the members of VLSI System Design Laboratory at Osaka University, especially, Ms. Akiko Fujii, Mr. Kazuki Yoshioka, Ms. Makiko Itoh, Mr. Takafumi Morifuji, Mr. Norimasa Ohtsuki, Mr. Jun-ichi Itoh, Mr. Yoshinori Jiyoudai, Mr. Eiichiro Shigehara, Mr. Katsuya Shinohara, Mr. Takeo Imai and Mr. Kiyou Shiyu.

The author would like to express his special thanks to Mr. Nobuyuki Hikichi from Software Research Associates, Inc., Dr. Tokinori Kozawa from STARC (Semiconductor Technology Academic Research Center), Dr. Masao Hotta from Hitachi Limited, Mr. Michiaki Muraoka from Matsushita Electric Industry Co., and the

members of LSI operation of Soliton Systems, K. K.

The author also thanks to the Professors and Specialists for helpful discussions and encouragement, especially, Prof. Hiroto Yasuura from Kyusyu University, Prof. Isao Shirakawa from Osaka University, Prof. Nagisa Ishiura from Osaka University, Prof. Ken-ichi Abe from Tohoku Univesity, Prof. Shunichi Kono from College of Science and Technology Tohoku, President Jiro Shimizu from Yamagata College of Industry and Technology, Prof. Toshiro Akino from Kinki University, Prof. Akihiko Yamada from Tokyo Metropolitan University, Prof. Hitoshi Kiya from Tokyo Metropolitan University, Prof. Yukihiro Nakamura from Kyoto University, Dr. Hiroaki Akaboshi from NEC, Mr. Hiroyuki Tomiyama from Kyushu University, Mr. Akihiko Inoue from Kyusyu University, and Mr. Hiroyuki Kanbara from ASTEM.

The author would like to thank Software Research Associates, Inc., Synopsys Co., Toshiba Co., STARC, NTT Communication Science Laboratory, Inc., Mentor Graphics Japan Co., Soliton Systems, K. K., and VLSI Technology, Inc., for their kind assistance to this work. This research was supported in part by STARC.

The author also would like to express his thanks to President Mitsuo Abe, Prof. Yutaka Togashi, Prof. Sigemasa Tsuchida, Prof. Kouich Fukushi, Prof. Tsuneo Kikuchi, Prof. Kouichi Fujimoto, Prof. Hideaki Sato, Prof. Hideki Noji, to the staffs and the students of Department of Electrical Engineering, Tsuruoka National College of Technology.

Finally, the author's thanks go to his family (his wife Setsuko Sato, daughter Ayaka Sato, son Kazusa Sato, son Takashi Sato, and relatives) for their help and encouragement.

Chapter 1

Introduction

1.1 Background

The integration density of VLSI (Very Large Scale Integration) is steadily increasing every year, and the performance of VLSI is also improving. In the year 2012, feature size (design rule), highest clock frequency and density on microprocessor chip are predicted as 0.05μm, 10GHz and 1.4B transistors, respectively [1]. For example, high clock frequency microprocessors are already developed or released [2] [3] [4] [5] [6], and related technologies to achieve 1GHz clock operation for microprocessor are being developed[7][8][9][10]. It is already possible to integrate a processor core and memories, such as DRAM and SRAM on the same chip [11]. Then, an Application Specific Integrated Processor (ASIP) [12], which contains a CPU core and peripheral circuits, will also include a large amount of memory block (for instruction and data) on the same chip by using Deep Submicron Technology (DSM). Throughout this thesis, "ASIP" is defined as a Systems on a Chip (SOC) that contains a CPU core, memory, and peripheral circuits.

ASIPs can be very effective when applied to specific domains of application such as communication, digital signal processing, image processing, mechatronics control, etc. The advantages of ASIPs are as follows:

(1) Better Cost-Performance.

(2) Design Flexibility.

(3) Higher Reliability.

ASIP can be used effectively in specific applications, such as embedded systems.

ASIP consists of highly functional blocks such as CPU, memory and peripherals, where conventional ASIC (Application Specific Integrated Circuit) can be viewed as one of peripherals of an ASIP.

One of the most distinguished features of ASIPs, compared to general-purpose processor is that their instruction set are sometimes originally designed for specific application domains in order to maximize the performance under the constraints of chip area, performance, or power consumption. Secondly, on-chip memories often play an important role in ASIPs. Where wide bandwidth and short access time of these on-chip memories enhance the performance of ASIPs considerably.

Other important features of ASIPs, compared to general-purpose microprocessors, are that typical ASIPs are fabricated in smaller volume and the expected design TAT (Turn Around Time) is much shorter than general-purpose microprocessor.

Considering these features, processor architecture design methodology based on hardware/software codesign method, which takes on-chip memory into account, would be most suitable for designing ASIPs.

1.2 Design Methodology for ASIP Development

1.2.1 CPU Core Design

There are two possible approaches to develop a CPU core in ASIPs.

1. Reuse an existing CPU core [13] [14] [15] [16].
2. Modify an existing CPU core [17].
3. Design a dedicated CPU core by modifying an existing CPU core or from scratch [18] [19] [20].

In the "general-purpose CPU core approach," predesigned application program development tools can be used. However, a general-purpose CPU core is not always most efficient in specific applications because the instruction set and hardware resources (such as register size, etc.) may not fit given application programs. Moreover, it is very difficult for application system designer to improve the design of the general-purpose CPU, because abundant experiences and deep knowledge are

necessary for tuning the CPU architecture, and the redesign of application program development tools are needed.

In the "modified CPU core approach," a designer can change the architectural papameters of existing CPU core to fit the target application, if the CPU core description is given in an HDL (Hardware Description Language) and the designer has a skill to improve the design using logic synthesis tools. This approach is based on the concept of design reuse. But, generally, the detail CPU design is not always disclosed as in an HDL description. Even if a synthesizable HDL design is available, the license fee of the CPU design is very expensive. And specific compiler for designed CPU core is needed as well as the dedicated CPU core approach, when the instruction set architecture is changed to improve the performance.

In the "dedicated CPU core approach," the CPU core that best fits the target application or application domain under design constraints, such as performance, hardware cost, power consumption, etc., can be provided by tuning the architectural parameters. It is possible to design a CPU core suitable for the application or application domain compared to a general-purpose CPUs. However, it is necessary to establish following technologies.

1. Optimization of instruction set and hardware architecture for given application programs.

2. Generation of synthesizable CPU core design.

3. Generation of application program development tools such as compiler, simulator and debugger.

The instruction set optimization problems is studied in Refs. [21] [22] [23]. Datapath synthesis and control logic synthesis techniques can be used to perform the CPU core design generation. Application program development tools generation problems are studied in Refs. [26] [20] [27] [28]. By integrating these technologies, the "dedicated CPU core approach" would be able to generate more efficient CPU cores than the "modified CPU core approach".

1.2.2 On-chip Memory System Design

Because the clock frequency for CPU core is usually higher than the access cycle frequency of on-chip memories and is much higher than that of off-chip memories, fast cache memories are required between CPU core and on-chip or off-chip memories to adjust the frequency gap [29]. Furthermore, embedded systems for specific application domain, such as multimedia processing, require huge memory space, additional external memory would be indispensable to realize such systems efficiently.

Required memory size of instructions and working data set strongly depend on the feature of the target application. Generally, the larger the portion of instructions and data are stored in the on-chip memory, the higher the system performance will become. Hence, an efficient memory system, which is able to manage several kinds of large on-chip memory and off-chip memory, is required.

1.3 Organization of the Thesis

This thesis is organized as follows.

In chapter 2, Flexible Servo Control Processor is introduced as a design case study of ASIP including data memory.

In chapter 3, the aim and goal of the PEAS project are described, where the requirements to ideal ASIP development environment are discussed. Then, the assumptions and restrictions of the PEAS-I system are described. Next, the outline of the implementation of the PEAS-I system is described. Finally, some experimental results are illustrated.

In chapter 4, a performance optimization method is proposed for hierarchical memory system in ASIPs, which consists of on-chip fast cache memory, a large amount of on-chip ordinary memory, and a huge off-chip memory. The performance of ASIP using the hierarchical on-chip memory system is evaluated first. Then, experimental results of the performance optimization method includes hit-ratio prediction, write-back penalty prediction and average memory access time estimation are described.

Chapter 2

A Case Study of ASIP Development

In order to achieve required performance, an ASIP can include specific instruction set, on-chip memory and some peripherals. In this chapter, the usage of on-chip memory in ASIP and the benefit of on-chip memory are explained with Flexible Servo Motor Control Processor as an ASIP design example.

2.1 Background

Currently, many of servo motor control systems are manipulated by means of software. However, conventional general-purpose microprocessors are not sufficient for the application to servo motor control systems from several reasons.

One of the most serious problems of a conventional microprocessor based system, when applied to servo motor control systems, is that such microprocessor is not powerful enough to execute the feedback control algorithms in a reasonable time duration. Then, frequent memory accesses to read parameters and to store data from/to slower memory such as PROM and DRAM decrease the system performance.

On the contrary, if the dedicated processor and memory is on a chip, the Flexible Servo Control System (FSC)[30] can be effectively realized. In this system, various kinds of servomotors, such as DC, AC synchronous, and AC induction motors, can be controlled by the same kind of hardware with different control software.

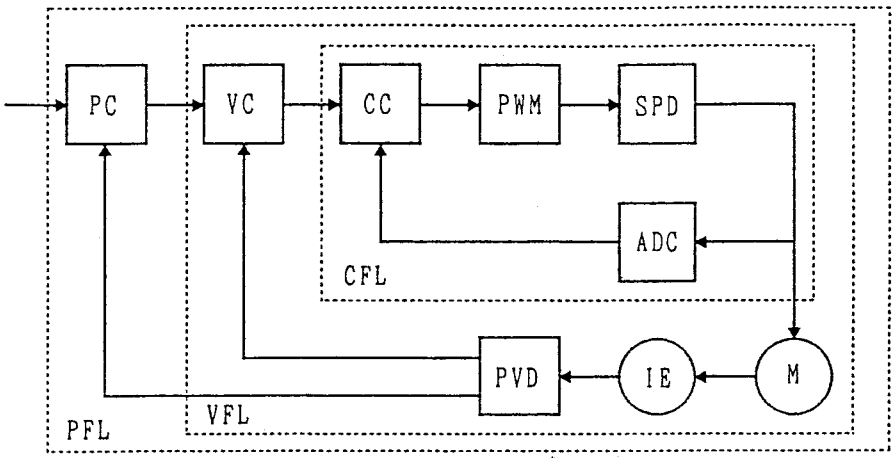
In this chapter, the requirements to the FSC are described first. Then, the

architecture of the FSP-3 (Flexible Servo motor control Processor - 3)[31][32] is proposed. Next, the implementation of the FSP-3 chip is illustrated. The FSP-3 has several advantages suitable for servo motor control applications.

2.2 Requirement to FSC

Generally, servo motor control algorithms consist of following feedback control procedures, as shown in Figure 2.1:

1. Current feedback control loop.
2. Velocity feedback control loop.
3. Position feedback control loop.



M : Servo motor
PC : Position control block
VC : Velocity control block
CC : Current control block
PWM: PWM signal generator
SPD: Servo pre-driver
ADC: A/D converter
PVD: Position/Velocity detecter
IE : Incremental encoder
PFL: Position feedback loop
VFL: Velocity feedback loop
CFL: Current feedback loop

Figure 2.1: Block Diagram of a Servo Motor Control System

Among them, current feedback control loop constructs the innermost loop, which should be executed as quickly as possible in order to reduce the response time of

the whole control system. A typical expected response time of this feedback control loop is about 25 microseconds. The calculation to be performed in this loop includes complicated operations such as coordinate transformation using trigonometrical functions and multiplication.

On the other hand, the expected response times for the velocity and position feedback control loops are, for example, several ten milliseconds and several hundred milliseconds respectively. While these response times depend on specific applications, these are typically much longer than the response time for the current feedback control loop. These two feedback control loops include multiplication and division operations to calculate complex control parameters. Theoretically, these three feedback control loops can be executed as communicating sequential tasks.

In summary, following functional requirements should be fulfilled to realize an efficient FSC:

1. Fast trigonometrical operations.
2. Fast arithmetic operation including multiplication and division operations.
3. Fast task switching.
4. Memory accesses Reduction.

2.3 The Architecture of FSP-3

2.3.1 Architecture Outline

The architecture of the FSP-3 is based on the following RISC (Reduced Instruction Set Computer) architecture with following features:

1. simple instruction set as shown in Table 2.1,
2. on-chip data memory and register file, which includes 16 bit 192 word data memory to store control parameters and 32 bit 29 word registers as accumulators,
3. two-stage pipelining.

Table 2.1: Instruction Set of the FSP-3

Mnemonic	Description
ADD	Addition
SUB	Subtraction
MUL	Signed Multiplication
DIV	Unsigned Division
SHR	Arithmetic Shift Right
SHL	Arithmetic Shift Left
AND	Logical AND
OR	Logical OR
NXOR	Logical Not Exclusive OR
MOVP	Move from PC Reg.
MOVU	Unsigned Move from Reg. to Reg.
MOVS	Signed Move from Reg. to Reg.
MOVM	Move between Mem. to Reg.
MOVI	Move Immediate Date to Reg.
JUMP	Jump Conditionally

Harvard Architecture, which physically split instruction and data buses, is adopted in order to broaden the bandwidth of instruction and data buses. The width of each instruction, data and address buses is 16 bit.

The main features of the FSP-3 architecture are as follows:

1. A RISC concept and Harvard Architecture are adopted.

2. 32 bit arithmetic operations can be executed in one instruction cycle.

3. Fast parameter read/write is available.

4. Special operations, such as multiplication, division and trigonometrical functions, are implemented as hardware.

5. A fast task switching mechanism is implemented.

6. Peripheral circuits, such as PWM (Pulse Width Modulation) signal generator and counters, are included.

7. Cast operation, such as 32 bit to 16 bit and 16 bit to 32 bit, can be performed by a register-to-register move operation.

8. The architecture is simple, comprehensive and easy to program.

The block diagram of the FSP-3 architecture is shown in Figure 2.2. The FSP-3 is constructed with the following six modules.

2.3.2 Control Module

The Control Module (CTRL_MOD) performs instruction fetch, instruction decode and external RAM access, as well as the control of other modules. The task switching control is also performed by this module.

2.3.3 Arithmetic and Logical Operation Module

The Arithmetic and Logical Operation Module (ALU_MOD) performs arithmetic and logical operations. This module contains an ALU, a barrel shifter, a multiplier and a divider. In order to perform trigonometrical operation, a sine look-up table is implemented as 16 bit 512 word ROM.

In order to accelerate the task switching, four sets of dedicated register files are included in this module. Each of three register files out of four is dedicated to one of three feedback control tasks respectively, and the other one can be used to execute another task such as interrupt handling. Each of these register files contains 32 bit 28 word registers.

2.3.4 RAM Module

In the RAM Module (RAM_MOD), 16 bit 192 word memory is supplied to store parameters; and 16 bit 64 word memory is also supplied to communicate with Peripheral Module and I/O Module.

2.3.5 Peripheral Module

The Peripheral Module (PEPRIPHERAL_MOD) includes a PWM signal generator. This module also includes counters used to detect the motor rotational position and

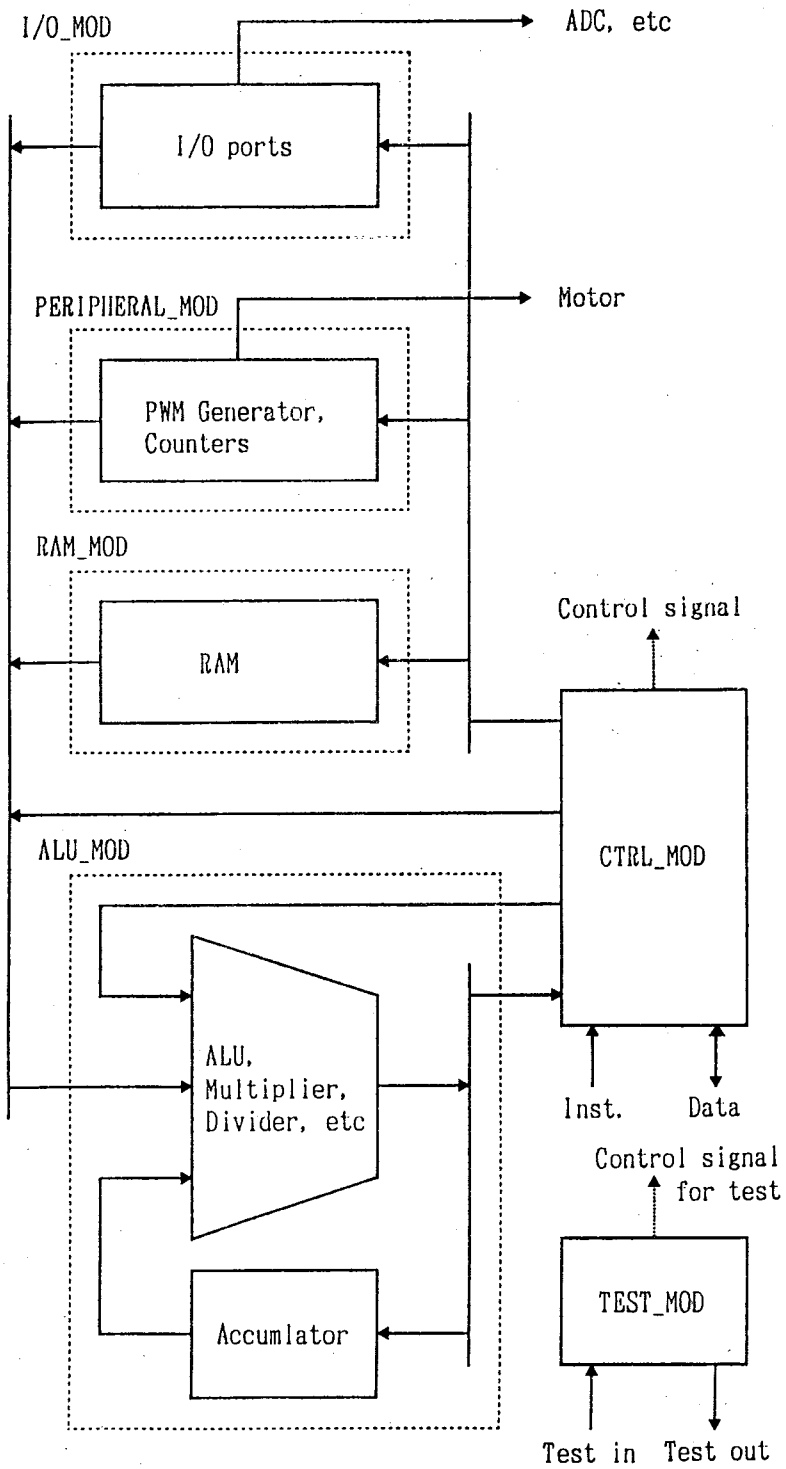


Figure 2.2: Block Diagram of the FSP-3 Architecture

the motor rotational velocity.

2.3.6 I/O Module

The I/O Module (I/O_MOD) contains three 16 bit I/O ports and several special purpose registers, which can be accessed from external devices. These registers are used for communication with other external processors and peripherals without synchronization.

2.3.7 Test Module

The Test Module (TEST_MOD) is used to detect the internal status and behavior of the FSP-3 from outside of the chip. When the "Test Mode" is selected, RAM and ROM can be accessed from outside of the chip, and the data values on internal buses can also be monitored from outside of the chip.

2.4 Design Results of the FSP-3

The design results of the FSP-3 are summarized in Table 2.2; and the chip layout is shown in Figure 2.3. The FSP-3 is able to execute most instructions in $100ns$. The FSP-3 has been designed using commercial silicon compiler GENESIL. The chip was fabricated by using 1.0μm CMOS technology from Toshiba Co. While the FSP-3 consists of 143,230 transistors, the design time of the FSP-3 was only about six man-months due to the use of the silicon compiler. The details of the design time are shown in Table 2.3.

The dedicated instructions reduce the program code size and improve the performance of FSP-3. And, as the on-chip data memory in FSP-3 reduces memory accesses to slower memory, the total system performance is improved. Then, the reduction of external memory access can achieve low power consumption of the system.

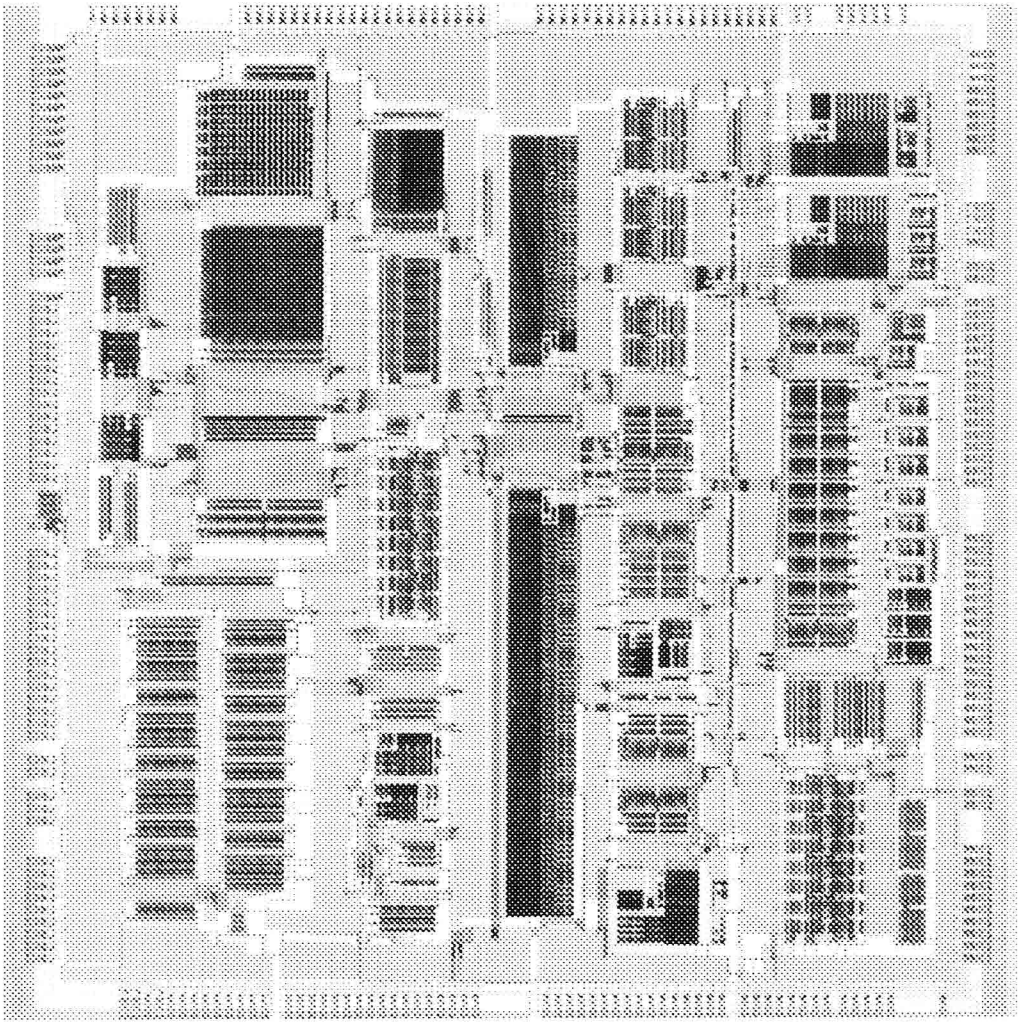


Figure 2.3: The FSP-3 Chip Layout

Table 2.2: Design Results of the FSP-3

Design time	1.0μ CMOS
Clock Frequency	10 MHz
Transistor Count	143,230 Tr.
Number of Basic Instruction	15
Instruction Execution Cycle	2 clocks
Pipeline Stage	2 stage
Power Consumption (Est.)	2.9 W
Chip Size	$13.0 \times 12.7\ mm^2$
Package Type	299 pin PGA
Design Tool	GENESIL
Design time	6 man-months

Table 2.3: Design Time of the FSP-3

Work	Design Time
Architectural Design	1 month
Logic Entry, Module Generation and Floorplaning (on GENESIL)	4 months
Design Verification including Test Vector Generation (on GENESIL)	1 month

2.5 Consideration of ASIP Development

FSP-3, which is an example of ASIP include dedicated CPU core, memory and peripherals, is effective system LSI to implement servo control applications. But throughout the development of FSP-3 by "dedicated CPU core approach," there are following problems should be solved in the ASIP development.

1. System architecture optimization

   The performance of ASIP can be improved by considering trade-offs of the amount of hardware of the CPU core, memory, peripherals and so on. Moreover, the CPU architecture optimization for the target application domain is the most important issue.

2. Software development

   An ASIP is the complex system which contains hardware and software modules. Then, in order to develop the ASIP in short TAT, optimal hardware and software modules automatical generation are required. Especially, the optimal software generation for specific CPU core is difficult without dedicated software development tools.

According to the development of FSP-3, integrated design method and environment for ASIP development is required.

Chapter 3

ASIP Development Environment

In this chapter, a hardware/software codesign system for ASIP development PEAS-I (Practical Environment for ASIP Development - type I) is described. PEAS-I system can decide an optimum instruction set and its implementation method under the given design constraints taking advantage of analysis results of application programs. Thus, PEAS-I system can reduce the execution time of application programs.

The efficiency and effectiveness of PEAS-I system were confirmed through several experiments. According to the experimental results, PEAS-I system gives accurate estimation of the chip area and performance of ASIPs before the detailed hardware design is completed. The experimental results also show that PEAS-I system is able to generate both hardware design and a set of application program development tools for a typical size ASIP within several hours. Then, the effectiveness of the design methodology in PEAS-I system was demonstrated through realistic design examples.

3.1 Aim and Goal of the PEAS Project

Most of electronic systems consist of hardware components and software components. These systems have been partitioned into hardware and software components by architects, then designed in detail by hardware and software engineers. In order to achieve the highest performance design at the least hardware cost, it is neccessary to optimize the system at a higher level. However, the optimum solution of this problem is not obvious for most architects and designers. Therefore, so far this

problem has been solved by a talented architect based on his/her experience and intuition.

A hardware/software codesign is a method to design large and complex electronic systems, such as ASIP which include hardware and software components. The hardware/software codesign method consists of system design, hardware design and software design. Optimization problems of an ASIP design can be classified into three types as follows:

1. Highest performance ASIP design under hardware cost and power consumption constraints.

2. Least hardware cost ASIP design under performance and power consumption constraints.

3. Lowest power dissipation ASIP under hardware cost and performance constraints.

In order to perform the system optimization, it is necessary to establish the following technologies:

1. Rapid prototyping: In order to estimate the performance, hardware cost and other design quality metrics of the target system at an early stage of the design process, the rapid prototyping at a high level of design abstraction is a key issue.

2. Accurate estimation: This is another key issue for codesign method to avoide miss-selection of architecture as well as to guarantee the optimality of the solution. There is a trade-off between design abstraction level and accuracy of the estimation. Therefore, it is necessary to develop a method to obtain an estimation as accurate as possible at a higher design level as possible.

3. Optimal partitioning: This is main issue to define hardware and software components of the system. The most important issue in solving optimization problems to achieve an optimal design of ASIP. There are three levels for partitioning process: process level, task/function level, and instruction level.

The aim of PEAS project is to investigate and establish an ideal environment for ASIP development. Several version of PEAS systems have been developed according to this aim, such as PEAS-I, PEAS-II and PEAS-III. Especially, PEAS-I system, which is the first implementation of PEAS concept, realizes following features:

1. Generate the highest performance ASIP hardware design as possible, where performance and resource cost estimations before detailed design completion would benefit the ASIP designers in making architectural decisions.

2. Generate software development tools for application programs simultaneously, which will also benefit the ASIP system designer if application program development tools are automatically generated.

In order to decide the optimum instruction set architecture on PEAS-I system, several optimization methods were implemented such as IMSP-1 (Instruction ser implementatin Method Selection Problem - type I)[22], IMSP-2[23], IMSP-3[24] and IMOP (Instruction set processor and Memory Optimization Problem)[25].

There are several issues to be considered. The first issue is the architecture model selection, that is, what kind of architecture is to be used for the given application. There are many candidate architectures which might be suitable for the given application programs. In the case of scalar architecture, the candidates would include, for example, RISC, CISC, and their variations. In the case of parallel architecture, there are superscaler and VLIW (Very Long Instruction Word) architectures to be considered. It is not obvious which kind of architecture would fit best to the given application and constraints.

The second issue is the detailed optimization of ASIP hardware. That is, what kind of and how many computing modules and memory modules including register file, are to be used in order to maximize the performance for the given application. Generally, it is not easy to predict the chip area, power consumption, and performance of an ASIP before the detailed design is completed. Therefore, it is very important to obtain as accurate estimations of these results as possible at an earlier stage of the design.

The third issue to be considered is how to provide a set of software tools for application program development. Most application programs are written in a high-level

programming language, such as C, Pascal or Fortran. When a high-level programming language is used to describe the algorithms in the application, then a compiler, an assembler, and a simulator are necessary to develop application programs. As already mentioned, it is not also an easy task to develop these software tools manually even if good software generation tools were available.

The goal of the first stage of the PEAS project was to solve the above mentioned three issues. That is, the current PEAS-I system goals are:

1. optimize the instruction set architecture for the given set of application programs and

2. generate application program development tools for the generated ASIPs.

3.2 Assumptions and Restrictions

Following assumptions were made to simplify the implementation of the system, because the main objective of this research is to demonstrate that the proposed method in this chapter is effective to generate both the ASIP hardware design and application program development tools automatically.

3.2.1 Target Applications

PEAS-I is assumed to synthesize ASIPs for specific application domains. General-purpose functionality is sacrificed for performance and efficiency. While the inputs to most high-level synthesis systems are structural or behavioral descriptions of the target architecture, PEAS-I system accepts a set of application programs written in C language and associated data set as input.

3.2.2 Design Optimization

The optimization is done in terms of instruction set selection and implementation. Maximizing the performance of an ASIP in executing the application programs under the constraints of chip area and power consumption is the primary goal of PEAS-I system. The secondary consideration is the efficient utilization of hardware resources.

3.2.3 Instruction Set Architecture

In the C language, operators and functions are treated apparently distinguished. In the design of the instruction set, to establish a new concept to treat them uniformly is needed. In the following description, "functionality" is defined as any of the operators or functions in the C language, independent of their implementation.

The functionality set can be divided into two subsets: **operators** and **functions**. The reasons are as follows:

1. It is easier for a C compiler to generate instructions corresponding to operators than to generate ones corresponding to functions.

2. While the set of operators can be clearly defined, the set of functions including user-defined ones cannot be given a priori.

Then, the set of operators is divided into two subsets: **primitive operators** and **basic operators**. The set of primitive operators is chosen so that any basic operators or functions can be realized by a series of primitive operators. Thus, the functionalities are divided into three classes as follows:

1. Primitive Functionalities

   The Primitive Functionalities (PF) can be realized by minimal hardware components such as ALU and shifter. The instruction set of the ASIP to be generated by PEAS system includes all the functionalities in PF.

2. Basic Functionalities

   The Basic Functionalities (BF) include the set of operators used in C language except those included in PF. Each functionality in BF can be implemented by one of the hardware modules, microprogram, or software subroutine.

3. Extended Functionalities

   The Extended Functionalities (XF) include those that correspond to library functions or user-defined functions. The functionalities in XF can be implemented by one of complex hardware modules, microprogram, or software subroutine. The hardware modules could be constructed as co-processors or peripheral modules.

3.2.4 Hardware Model

Only the CPU core of the ASIP is to be generated in PEAS-I. The instruction set architecture model is based on the abstract machine model of the GNU C Compiler (GCC) [33]. In this model, the full set of instructions that can be generated by GCC is treated as a base for the instruction set architecture generator. According to the classification of the functionalities which is described below, the full set of instructions is divided into three categories:

- Primitive RTL (PRTL)
- RTL (BRTL)
- Extended RTL (XRTL)

Where RTL stands for the Register Transfer Language, which is the intermediate language of GCC. The PRTL is always included in the ASIP instruction set as a core. Any C program can be executed, using PRTL. The BRTL corresponds to other operators in C language, and the XRTL corresponds to pre-defined or user-defined functions. The BRTL and XRTL are optional.

The PRTL can be implemented by minimum hardware components and some control logics. The BRTL and XRTL can be implemented in several ways. They could be implemented by hardware modules, microprogram, software subroutines using only the PRTL and part of the BRTL or XRTL. The minimum portion of the generated CPU core is called the "kernel" which consists of an ALU, a 1bit shifter and a register file. The kernel coressponds to Primitive operations.

The CPU core may include other functional units (FUs) auch as multiplier, divider, barrel shifter and so on. Then, these FUs coresspond to Basic operations and Extended operations.

The pipeline stages of the CPU core consist of following four stages:

(1) IF (instruction fetch and decode),

(2) EX (execution),

(3) MEM (memory access), and

(4) WR (write back to register).

While each of IF, MEM and WR stages takes only one cycle, EX stage takes one or more cycles.

Table 3.1 shows the PRTL, BRTL and XRTL that can be generated by PEAS-I system.

Table 3.1: PRTL, BRTL and XRTL

class	category	functionality
PRTL	arithmetic	add, sub, and, ior, xor, one_compl, neg, ashl1, ashr1, lshl1, lshr1
	transfer	mov, set
	control	jmp, nop, beq, bne, bgt, blt, bltu, bge, bgeu, bleu, bgtu, call, return
BRTL	arithmetic	mul, umul, div, mod, udiv, umod, trunc, extend, zero_extend, rotate, ashr, ashl, lshr, lshl
	transfer	movpc
XRTL	arithmetic	abs, sqrt, ffs, sin, cos, tan, *user-defined functions*, etc.

3.3 Implementation of PEAS-I system

PEAS-I system consists of four subsystems: the Application Program Analyzer (APA), Architecture Information Generator (AIG), CPU Core Generator (CCG) and Application Program Development Tool Generator (DTG). The configuration of PEAS-I is shown in Figure 3.1.

3.3.1 Application Program Analyzer (APA)

The APA profiles application programs with corresponding data sets. The output of the APA includes an execution frequency count of operators and functions used

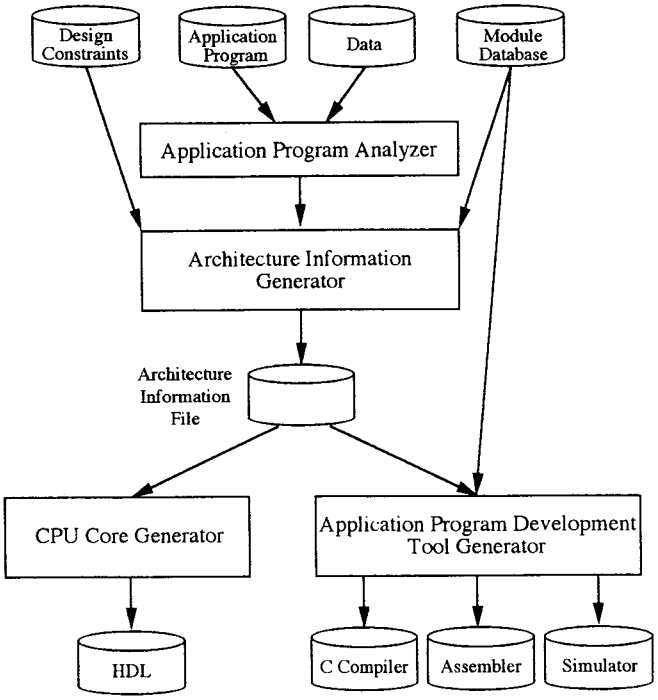


Figure 3.1: Configuration of PEAS-I System

in the application programs. The APA accepts application programs written in C language and associated data. The execution frequency count of functionality is measured by compiling the input programs and executing the programs with the simulator. When a set of application programs are given, the profiled results of each program are to be mixed with an appropriate ratio. A sample program and its analyzed results are shown in Figure 3.2 and Figure 3.3, respectively.

```
#define ABS(X) ((X < 0) ? (-X) : (X))
main()
{
    int i, x, y, z;
    x = 3; y = 12;
    for(i = 0; i < 10; i++) {
        x++; y++;
        x = psqrt(x, y);
    }
}
int psqrt(x, y)
int x, y;
{
    int a, b, result, dummy;
    x = ABS(x); y = ABS(y);
    if (x >= y) {
        a = x; b = y;
    } else {
        a = y; b = x;
    }
    result = ((a * 7) + (b * 7)) >> 2;
    dummy = result & 0x00000001;
    result = (result / 2) + dummy;
    if (a > result) result = a;
    return(result);
}
```

Figure 3.2: A Sample Input Program for APA

```
frequency
        kernel          704
        mulsi           10
        umulsi          0
        divsi           0
        udivsi          0
        modsi           0
        umodsi          0
        ashlsi          10
        ashrsi          20
        lshlsi          0
        lshrsi          0
        negsi           0
        cmpsi           0
        tstsi           0
        truncsihi       0
        truncsiqi       0
        extendhisi      0
        extendqisi      0
        z_extendhisi    0
        movpc           0
        rotlsi          0
        rotrsi          0
## Number of F'ty 744
## Execution time 904 clock
```

Figure 3.3: The Output Result of the APA

3.3.2 Architecture Information Generator (AIG)

The AIG accepts the profiled results from the APA, and decides the instruction set architecture of the ASIP. This task can be performed by tuning parameters in the architecture models for the instruction set and CPU core, so that the performance of the ASIP can be statistically maximized regarding the given application programs under the constraints of chip area and power consumption. An example of architecture information is shown in Figure 3.4.

```
#deifne        IBUS     32
#deifne        DBUS     32
#deifne        REG      8
#deifne        MOVPC    0
#deifne        MULT     1
#deifne        DIV      0
#deifne        SHIFT    1
#deifne        EXTEND   0
#deifne        TRUNC    0
#deifne        ROTATE   0
```

Figure 3.4: An Example of Architecture Information

Out of the full set of instructions which include BRTL and XRTL, the AIG selects an optimum instruction set that maximizes the performance of a given application program. PRTL with the selected subset of BRTL and XRTL implemented by hardware represents an optimum instruction set of the designed ASIP. This problem can be formalized as an integer-programming problem and can be solved, using a branch-and-bound method.

In order to decide the optimum instruction set, AIG uses a design method based on a combinatorial optimization technique which formulates the Instruction set implementation Method Selection Problem (IMSP). Three types of IMSP problems are solved by AIG to support PEAS-I system as follows:

1. IMSP-1 [22]: A design problem that considers the performance of the chip as the main factor in the design and tries to maximize it under the constraints of the chip area and power consumption. This problem does not take the

function module sharing among operations into consideration. IMSP-1 was formalized as an integer programming problem.

2. IMSP-2 [23]: A design problem that considers the performance of the chip as the main factor in the design and tries to maximize it under the constraints of the chip area and power consumption. This problem takes into consideration the functional module sharing among operations. IMSP-2 was formalized as a comninatorial problem.

3. IMSP-3 [24]: A design problem that considers the hardware resources as the main factor and tries to minimize them under the constraints of chip performance and power consumption. IMSP-3 was formalized as a comninatorial problem.

One of the advantages of using a formal optimization method to design the instruction set is that the performance prediction of the designed ASIP can be done before the completion of detailed design. This prediction feature is very effective in designing a high quality design of ASIP in a short TAT. Here, the unit of MFPS (Million Functionalities Per Second) is defined to measure the performance of ASIPs as follows:

$$MFPS = \frac{\sum_{i=1}^{n} f_i}{\sum_{i=1}^{n} f_i \times t_i} \times F \tag{3.1}$$

where

n denotes the total number of functionalities,

f_i denotes the execution frequency count of functionalities $\#i$,

t_i denotes the execution cycle of the modules that implements the functionalities $\#i$ and F denotes the clock frequency in MHz.

MFPS represents the normalized performance of the target CPU. In order to evaluate the performance of the CPU when the design constraints are changed, the estimation in MFPS is useful. When such estimation is performed, the total results of area and power consumption are obtained by calculating the sum of each value that is taken beforehand for each hardware module.

3.3.3 CPU Core Generator (CCG)

In PEAS-I, the generated architecture is based on the pipelined Harvard Architecture. In order to generate the CPU core design description easily, implementation method of each instruction, size of register file, and bus width are parameterized. In the following discussion, instruction- and data-bus widths are both fixed to 32 bit. Register count in the register file is assumed to be given by the designer. To facilitate the generation of the CPU core design, the method to add a necessary pre-designed hardware module for the framework of CPU core is adopted.

CCG generates the CPU core design in the form of an HDL using the architecture information obtained by the AIG. Then, the actual CPU core design can be obtained by compiling the HDL description using a logic synthesis tool. In the current implementation, SFL[34] description is used as the final output of CCG. The actual design (netlist description) can be synthesized by the PARTHENON system[35][36] from NTT.

3.3.4 Application Program Development Tool Generator (DTG)

DTG generates a set of application program development tools, which includes a C compiler, an assembler and a simulator. These tools run on a workstation to develop ASIPa and their application programs.

1. C compiler

   PEAS-I system generates a C compiler for the ASIP synthesized by PEAS-I. This task can be performed by assigning values to the parameters in the "md" file and the header files in GCC. The "md" file defines the instruction set architecture information such as instruction patterns, and the header files define other architecture information, such as bus width, register configuration, etc., of a target CPU. The C compiler generator of PEAS-I consists of the "md" file generator and the header file generator, which generates "md" and header files respectively, from the architecture information decided by AIG.

2. Assembler

   The assembler generated by PEAS-I system translates the intermediate lan-

guage (RTL) into the binary machine code for the synthesized ASIP CPU. The assembler is generated as a C program and is compiled because it should take the architecture information into account. It is possible to construct the assembler as a kind of interpreter, but a compiled program will be much more efficient.

While the GCC takes care of the code optimization, in the current implementation of PEAS-I, the generated C compiler only takes care of the intermediate instruction (RTL) generation and register assignment optimization. Then, the assembler translates the RTL code into machine code, where such RTL instructions that are not directly implemented by hardware modules are expanded into a sequence of directly implemented machine instructions.

3. Simulator

   A simulator is also generated by PEAS-I system, because the simulator should also take into account the architecture information such as the valid instructions and register count. The simulator is able to report the performance profile of the application program, such as total execution cycles, execution frequency of each instruction, etc.

   The simulator diagnoses the machine code generated by the assembler so that possible defects of the implementation of PEAS-I system can be recognized; this feature was found to be very effective in developing the whole PEAS-I system. These tools can also be used for tuning PEAS-I system as well as for developing the application programs of the target ASIP.

3.4 Preliminary Experiments

3.4.1 Objective

Several experiments have been performed to confirm the effectiveness and efficiency of PEAS-I system, which include the following observations:

1. Estimation accuracy of gate count and performance:

   Obtaining accurate estimations of gate count and performance before logic

synthesis is essential in achieving a high quality design in short TAT.

2. Architectural adaptability of the generated ASIP CPU:
   PEAS-I system should generate an appropriate architecture for the given application under the design constraints.

3. Computational efficiency:
   Short design TAT is quite important to develop ASIPs.

3.4.2 Sample Programs

The sample programs used in the experiments are as follows:

1. *NORM*: A program that calculates an approximate integer value of $\sqrt{x^2 + y^2}$, where x and y are both integers.

2. *FPE*: A floating point emulator program, which emulates floating point +, -, * and / operations using integer arithmetic and shift operations.

3. *GCD*: A program that calculates the GCD (Greatest Common Divisor) using Euclid's method.

These programs were fed to the APA to get the execution frequency count of each instruction. Parts of the execution frequencies of operators in these programs are shown in Table 3.2. In this table, "kernel" represents the minimum hardware modules, which consist of an ALU, a 1-bit shifter and some control logic to implement the PRTL mentioned in Section 3.2.4. The "mul", "ashl", "lshr" and "div" represent multiply. arithmetic left shift, logical right shift, and division, respectively.

3.4.3 Experiment

In this experiment, *NORM* program is used as an input application program. According to the execution frequency count shown in Table 3.2, a barrel-shifter and a hardware multiplier are known as hardware component candidates which could be included in the target CPU core. The specifications of part of the computing modules are shown in Table 3.3. Where "b_sht" denotes a barrel-shifter, and "mul_1",

Table 3.2: Relative Execution Frequency of each Operation in Sample Programs

Instruction	execution count (%)		
	FPE	*NORM*	*GCD*
PRTL Operations	95.1	94.4	93.9
multiplication	0.4	1.4	0.0
barrel arithmetic shift	3.7	4.2	0.0
barrel logical shift	0.8	0.0	0.0
division	0.0	0.0	6.1
others	0.0	0.0	0.0

"mul_17" and "mul_32" denote multipliers that execute a 32 × 32 bit multiplication in 1, 17 and 32 clock cycles, respectively. Accordingly, there are eight different effective CPU core designs (A through H) for this application as shown in Table 3.4.

Table 3.3: Specification of part of Computing Modules

module name	gate count	power (mW/MHz)	execution cycles	implied instruction
kernel	15809	59.88	1	PRTL
mul_1	7781	30.16	1	mul, umul
mul_17	3924	16.15	17	mul, umul
mul_32	2938	12.38	32	mul, umul
b_asht	852	3.37	1	ashr, ashl
b_lsht	844	3.32	1	lshr, lshl
b_alsht	855	3.82	1	ashr, ashl, lshr, lshl
div_19	7095	28.25	19	div, udiv, mod, umod
div_35	5334	21.50	35	div, udiv, mod, umod

3.4.4 Estimation Accuracy

Using the *NORM* sample program, the gate count and performance of each of the CPU core design A through H in Table 3.4 were estimated and shown in Table 3.5. Power consumption constraints were ignored to simplify the experiment combina-

Table 3.4: Eight Possible CPU Core designs for *NORM* Application Program

design	constraint
A	kernel
B	kernel + b_sht
C	kernel + mul_32
D	kernel + mul_17
E	kernel + mul_1
F	kernel + b_sht + mul_32
G	kernel + b_sht + mul_17
H	kernel + b_sht + mul_1

tions. These designs were synthesized, using PARTHENON and cell library VTI.lib from VLSI Technology Inc. Then, their execution cycles were measured, using the simulator.

1. Gate Count Estimation

   Part of the estimated gate count is compared with the measured values in Table 3.5. From this table, each estimated gate count was found to be very accurate – at most 3.4% different from the measured value. This is because the estimation is performed using already optimized modules.

2. Performance Estimation

   The estimated and measured performances are shown in Table 3.6. The estimated performances were found to be fairly accurate: at most 25% different from the measured value. It is also observed that the more the functionalities are implemented by hardware modules, the more accurate the estimation becomes. The reason behind this phenomenon can be interpreted as follows: when a BRTL or an XRTL operation is implemented by software subroutine, the execution cycle of the instruction can distribute on a certain range. Note that the execution cycle of a PRTL is a constant, because it is implemented by hardware module.

Table 3.5: Comparison of Estimated and Actual Gate Count

design	(a)	(b)	error (%)
A	15809	15809	0.0
B	16653	16678	0.11
C	16661	16685	0.14
D	16664	16700	0.21
E	19599	19218	1.90
F	19602	19221	1.98
G	20588	20121	2.32
H	24445	24707	1.06

(a) estimation (b) measurement

Table 3.6: Comparison of Estimated and Measured Performance

design	(a)	(b)	error (%)
A	4.04	4.61	12.3
B	4.40	5.23	15.0
C	6.41	6.54	13.0
D	7.37	7.87	6.3
E	7.95	7.27	9.0
F	8.99	8.99	0.0
G	9.45	9.45	0.0
H	10.0	10.0	0.0

(a) estimated average (b) measurement

3.4.5 Architectural Adaptability

All three samples to confirm the adaptability of PEAS-I system are designed. Figure 3.5 summarizes the performances of *NORM*, *FPE* and *GCD* sample programs for different gate count constraints. From this figure, it is known that PEAS-I system adaptively generates ASIP CPU cores according to the features of applications. That is:

1. The more the constraint is relaxed, the better the performance of the ASIP becomes.

2. PEAS-I system generates different architectures for different applications even

if gate count constraints are the same.

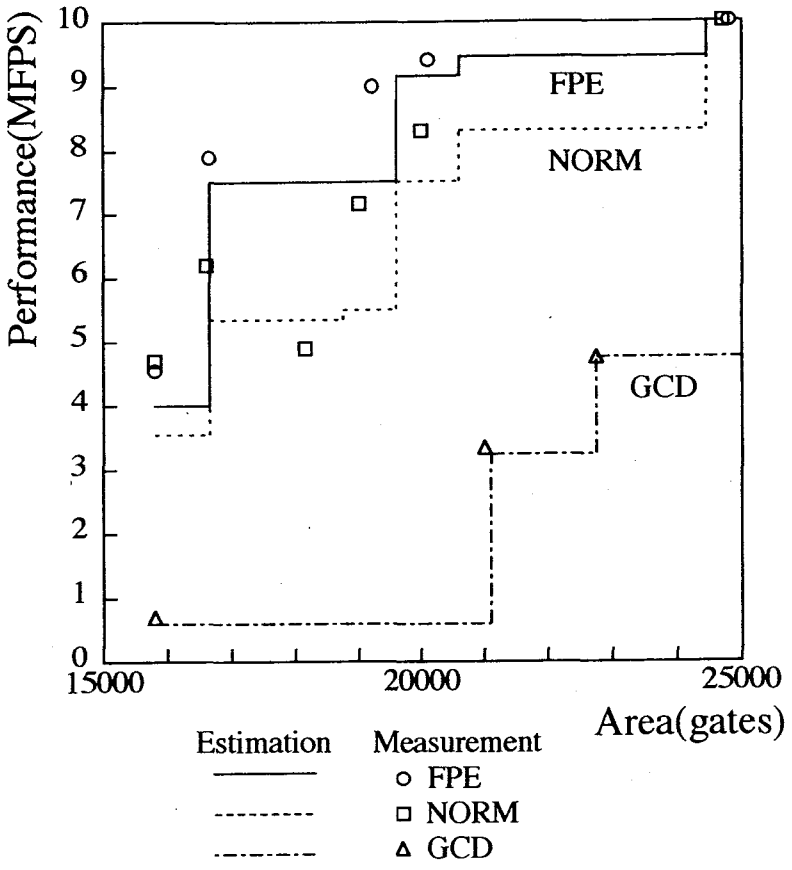


Figure 3.5: Comparison of Performance Estimation and Measurement

3.4.6 Efficiency

The efficiency of PEAS-I system can be evaluated by measuring the total computation time to generate the HDL description of an ASIP and to generate application program development tools. PEAS-I system was found to be very efficient. For example, typical computation times for APA, AIG, CCG and DTG to generate a 20K gate ASIP CPU and its software tools were a few minutes, 1 second, 3 hours and 20 minutes, respectively, on an NS SUN4/40 (about 15.8 MIPS as fast). Thus, the total

computation time of PEAS-I system to synthesize the 20K gate ASIP hardware and software tools would be at most 3.5 hours, which is remarkably short compared to conventional ASIP design methodologies.

3.5 Effectiveness Evaluation

3.5.1 Experiments

Five sample programs have been used in the experiment as shown in Table 3.7. The input/output data and internal operations of the sample programs are both of integer type. The features of general-purpose CPUs for this experiment are shown in Table 3.8. The general-purpose CPUs are classified into three classes:

1. Old generation (non-pipelined scaler processor), which includes Motorola MC68030 and Intel 386SX,

2. New generation (pipelined scalar processor), which includes Intel 486DX, MIPS R3000, etc.,

3. Latest generation (super-scaler and super-pipeline proceossor), which includes Intel Pentium, Sun Microsystems SuperSPARC, etc.

Table 3.7: Sample Programs

Sample	Description
ADP	ADPCM Decode/Encode (256 data)
DCT	Discrete Cosine Transform (8 points)
FFT	Fast Fourier Transform (128 points)
HUF	Huffman Decoding (100 char.)
PAR	PARCOR Filter (128 data)

The execution time of Intel 386SX, IBM 486SLC2, Intel 486DX and Intel 486DX2 has been measured using DOS Extender (GO32). The execution time of other general-purpose CPUs has been measured on workstations when the load average was low. The compiler used in this experiment was GCC except for MIPS R3000

Table 3.8: General-Purpose CPUs

CPU	Clock Freq. [MHz]	Cache Size [Kbytes]		Used machine
		Primary	Secondary	
PEAS	16	0	0	Simulator
MC68030	20	0.5	0	Fujitsu S-3/80
i386SX	16	0	0	Toshiba J3100
i486DX	33	8	128	IBM PS/V
i486DX2	66	8	128	Gateway P4D-66
IBM486SLC2	66	16	64	SusTeen WinMaster66i
R3000	32	0	128	Kubota Titan750
R6000	60	80	512	MIPS RC6280
SPARC	40	0	64	Fujitsu S-4/2
microSPARC	50	6	0	Fujitsu S-4/LX
Pentium	60	16	256	Gateway P5-60
SuperSPARC	40	36	0	Sun SPARCStation10
R4000PC	80	16	0	NEC EWS4800/310
R4400PC	133	32	0	NEC EWS4800/330
PA-RISC	66	0	384	HP9000/730
Alpha21064	150	16	256	DEC3000/300

and DEC Alpha21064. For latter CPUs, CC has been used because GCCs for these CPUs were not available.

PEAS-I system generates the CPU core under the following design conditions: (1) target clock frequency is 16MHz and (2) Cell library is VSC470 (0.8μm, CMOS). When a CPU core requires multiplication and/or division instructions, a multiplier which executes 32-bit signed-fixed-point multiplication in 1 cycle, and a divider which executes 32-bit signed-fixed-point division in 17 cycles are used. The execution time of PEAS-I CPUs was measured by using a simulator. The simulator reports the execution time including the pipeline interlock delay.

3.5.2 Results

The execution frequencies of functionalities of each sample program are shown in Table 3.9. Where "mul", "div", "shift" and "extend" correspond to multiplication, division, barrel-shift and sign extension instructions respectively. And "kernel" corresponds to primitive instructions such as load, store, branch, etc. Table 3.10 shows the specification of PEAS-I CPUs. Table 3.11 compares the execution time of sample programs on a PEAS-I CPU and general-purpose CPUs.

Table 3.9: Execution Frequency of Functionalities (unit[%])

Category	ADP	DCT	FFT	HUF	PAR
kernel	89.4	85.8	83.6	89.5	74.2
mul	0.0	7.1	8.0	0.0	18.9
div	0.0	0.0	0.0	0.0	3.1
shift	9.3	7.1	8.4	9.6	3.5
extend	1.3	0.0	0.0	0.9	0.3

The following observations are obtained from the experiment results.

1. Comparison to old generation processors:

   Because the PEAS-I CPU performs pipeline processing, it might be obvious that PEAS-I CPU is more efficient than old generation processor. However, the comparison is significant because these processors are often used for embedded applications. Throughout the samples, the execution time of the PEAS-I CPU

Table 3.10: Specification of PEAS-I CPU

Sample	Reg. Count	Exec. Cycle	Gate Count
ADP	19	36610	21214
DCT	22	123	30204
FFT	28	26574	32958
HUF	19	81727	21214
PAR	20	69744	35170

Table 3.11: Experiment Results

CPU	PAR		FFT		HUF		ADP		DCT	
	(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)
PEAS	4.4	1	1.7	1	5.1	1	2.3	1	7.7	1
MC68030	48.7	0.09	18.0	0.09	16.0	0.32	5.6	0.41	64.0	0.12
i386SX	33.1	0.13	16.3	0.10	31.1	0.16	15.6	0.15	72.0	0.11
i486DX	8.4	0.52	3.0	0.57	3.1	1.65	1.5	1.53	5.4	1.43
i486DX2	4.1	1.07	1.2	1.42	1.9	2.68	0.77	2.99	2.9	2.66
IBM486SLC2	4.6	0.96	1.6	1.06	3.3	1.55	1.4	1.64	5.0	1.54
R3000	5.6	0.79	1.2	1.42	2.4	2.13	0.88	2.61	5.4	1.43
R6000	4.5	0.98	0.91	1.87	1.7	3.00	0.57	4.04	1.2	6.42
SPARC	14.1	0.31	3.5	0.49	2.5	2.04	0.71	3.24	8.7	0.89
microSPARC	11.0	0.40	2.7	0.63	2.9	1.76	0.61	3.77	4.5	1.71
Pentium	2.6	1.69	0.65	2.62	1.0	5.10	0.69	3.33	2.3	3.35
SuperSPARC	10.3	0.43	2.0	0.85	1.7	3.00	0.46	5.00	1.1	7.00
R4000PC	3.2	1.38	0.61	2.79	1.2	4.25	0.44	5.23	0.76	10.1
R4400PC	1.9	2.32	0.36	4.72	0.70	7.29	0.28	8.21	0.44	17.5
PA-RISC	3.7	1.19	0.94	1.81	1.3	3.92	0.52	4.42	2.8	2.75
Alpha21064	2.5	1.75	0.31	5.48	0.60	8.50	0.23	10.0	0.72	10.7

(a) Execution Time [ms],
(b) Performance Ratio (Perf. of PEAS-I CPU/Perf. of general-purpose CPU)

is about 70% to 90% shorter than those CPUs i.e., PEAS-I CPU is 3 to 10 times as efficient as those CPUs. Moreover, the amount of hardware of the PEAS-I CPU is much smaller than those of CPUs.

2. Comparison to new generation (pipelined) processors:
   The execution time of PARCOR filter on the PEAS-I CPU is shorter than new generation CPUs. The reason is mainly due to the efficiencies of multiplication and division in PEAS-I CPU. The execution time of other samples programs on the PEAS-I CPU is roughly as same as that of new general-purpose CPUs.

3. Comparison to the latest generation processors (superscalar and super-pipeline)
   The execution time of PARCOR filter on PEAS-I CPU is almost as same as that of these processors. This is due to the difference of the execution time of multiplication and division instructions. Especially, SuperSPARC dose not has multiplication and division instructions, the execution time of PARCOR filter and FFT is longer than PEAS-I CPU. The execution time of other sample programs on the PEAS-I CPU is up to 17 times of those of these processors.

The general-purpose CPUs used in this experiment do not have high-speed hardware multiplier or divider. Therefore, these CPUs are not suitable for execution of application programs with high execution frequencies of multiplication and division.

Chapter 4

On-Chip Memory System for ASIPs

In this chapter, a performance optimization method is proposed for hierarchical memory system in ASIPs, which consists of on-chip fast cache memory, a large amount of on-chip ordinary memory, and a huge off-chip memory. Using the hierarchical on-chip memory system, the performance of an ASIP could be improved up to 50 % compared to that with conventional cache memory system.

The performance optimization method includes **hit-ratio prediction**, **write-back prediction** and **average memory access time estimation**. From the experimental results, it is known that the proposed method can decide an optimal configuration of the on-chip memory much more efficiently than conventional optimization methods based on the iteration of cache simulation. The proposed method can estimate the average memory access cycle very accurately for fully associative caches. Even when the cache memory is non-fully associative, the performance of the on-chip memory configurations obtained by the proposed method was found to degrade only up to 5 % compared to that by the conventional cache simulation method.

4.1 Hierarchical On-Chip Memory System

The configuration of on-chip memory system is categorized as follows:

1. Cache Only Configuration (COC)

2. Ordinary Memory Configuration (OMC)

3. Composite Memory Configuration (CMC)

The COC consists of fast cache memory only. The OMC consists of ordinary memory such as SRAM and DRAM, without cache memory. The CMC consists of both cache memory and ordinary memory. The configuration of CMC includes following components:

1. cache memory,

2. ordinary memory,

3. external (off-chip) memory, and

4. TLB (Translation Look-aside Buffer),

where internal (on-chip) memory consists of cache memory and ordinal memory. The features of this memory system can be summarized as follows.

1. Large amount of on-chip storage with access control logic is included.

2. On-chip cache memory works as a buffer among internal functional unit (such as CPU core), on-chip ordinary memory such as DRAM and external (off-chip) memory.

The organization parameters for each component of CMC are as follows:

1. Cache Memory

   - Block Size
   - Number of Blocks
   - Degree of Associativity

2. Ordinary Memory

   - Bit Width
   - Number of Words

3. External Memory

   - Bit Width
   - Number of Words

4. TLB

   - Number of Entries

4.2 Modeling the On-Chip Memory System

4.2.1 Hardware Cost and Performance Model

The hardware cost and the performance can be estimated by using the parameters described in the previous section. Hardware cost can be estimated as follows:

$$\begin{aligned} Hardware\_Cost &= Cache\_Cost + Internal\_Mem\_Cost \\ &+ TLB\_Cost + External\_Mem\_Cost \qquad (4.1) \\ &= h_1 \times (Block\_Size \times Number\_of\_Word) \\ &+ h_2 \times Associativity \\ &+ r_1 \times (Bit\_Width \times Number\_of\_Word) \\ &+ r_2 \times (log(Number\_of\_Word)) \\ &+ t \times Number\_of\_Entry \\ &+ e \times (Bit\_Width \times Number\_of\_Word) \qquad (4.2) \end{aligned}$$

where the dimension of the coefficients h_1, h_2, r_1, r_2, t_1 and e_1 is area per unit. Then, the performance can be estimated as follows;

$$C = C_1 + C_2 + C_3 \qquad (4.3)$$

$$T = k_1 \times C_1 + k_2 \times C_2 + k_3 \times C_3 \qquad (4.4)$$

where C is the total memory access count and T is the total memory access time. C_1, C_2 and C_3 are memory access count of cache memory, internal memory and external memory, respectively. Coefficients k_1, k_2 and k_3 denote access times for cache memory, ordinary memory and external memory, respectively.

4.2.2 Assumptions

In order to evaluate the effectiveness of the memory system configuration, the experiment, described in the next section, was carried out under following assumptions.

1. The hit-ratio of the instruction cache is high enough to ignore the effect of instruction cache miss-hits. Because the program code size is small in most embedded applications so that all instructions can be stored in the cache.

2. Both replacement strategies of cache and ordinary memory are based on an LRU (Least Recently Used) method. In the experiment described in the next section, the LRU algorithm is assumed to be an ideal one.

3. The hardware cost of external memory is not counted, because the experiment targets on the memory trade-off on a chip.

4.2.3 Preliminary Analysis

Let us consider the behavior of the system model. In the following discussion, the hardware cost of the internal memory is fixed at constant K. In general, the larger the cache size becomes, the higher hit-ratio can be expected and the average memory access time can be reduced. However, as the proportion of cache memory increases, the hit-ratio of ordinary memory becomes lower and the total miss-hit penalty increases. Therefore, the optimal proportion of cache (S_{opt}) can be decided by considering above-mentioned effects, where S_{opt} will be the optimal proportion of cache. Please note that (1) the unit hardware costs of cache and ordinary memory are different, and (2) S_{opt} depends on the sum of hardware costs of both cache and ordinary memory.

4.3 Exffectiveness of Hierarchical Memory System

4.3.1 Experiment

In this section, the performance trade-off between the total memory access time and the proportion of cache under given hardware cost constraint was examined

using a simplified model of the memory system. In this experiment, following key information has been investigated to estimate the hit-ratios of cache and ordinary memory:

1. total access counts to cache memory,
2. total access count to ordinary memory, and
3. total access count to external memory.

Target core processor was assumed to be DLX[29]. A DLX simulator and its cross C compiler[37] were used to obtain the bus trace information for this experiment. This trace information was analyzed by **dinero cache simulator**[38]. Sample programs used in this experiment were PVRG-JPEG[39], PVRG-MPEG[40] and PVRG-P64(ITU-T H.264)[41] programs. The parameters used in this experiment were fixed as follows:

1. Let both block sizes of cache and bit width of ordinary memory be 256 bit each,
2. Let bit width of external memory be 64 bit,
3. Let association method be a direct-mapping,
4. Let data replacement method be write-back,
5. Let memory access time k_1, k_2 and k_3 be 1, 5 and 40, respectively, and
6. Let the proportion of hardware costs (area coefficients) r_1 of internal memory to h_1 of cache be 1:10.

4.3.2 Experimental Results

Figure 4.1 through 4.6 show the results of memory access time v.s. the proportion of the cache memory size, where S is the proportion of cache in the given hardware cost constraint of internal memory size, then "X K" designates the hardware cost constraint which means the hardware cost of X Kbytes cache. Figure 4.7 through 4.12 show the speedup of the memory configuration compared to the cache only

configuration (COC). The speedup is defined by Eq. (4.5), where $T(X)$ is the total memory access time when proportion of cache is X %.

$$Speedup = \frac{T(100) - T(X)}{T(100)} \tag{4.5}$$

In Eq. (4.5), $T(100)$ is the total memory access time when COC was used.

4.3.3 Consideration

Experimental results can be summarized as follows.

(1) Execution cycles can be reduced for the same amount of internal memory by using hierarchical memory configuration.

(2) Hierarchical memory configuration could reduce internal memory area.

(3) Proportion of cache memory in CMC must be decided carefully, taking into account the size of working data set of target applications.

These results can be interpreted as follows:

(1) Performance Improvement

Figure 4.7 through 4.12 show the improvements of execution cycles when the proportion of cache to the amount of internal memory of CMC was varied. In this experiment, the optimal proportion of cache (S_{opt}) dropped between 25% and 50%. The hierarchical memory configuration achieved up to 50% improvement compared to a traditional cache only memory configuration (COC) in execution cycles. The improvement ratio under tight hardware cost constraint (that is, under tight internal memory area constraint) was generally larger than that under loose hardware cost constraint. However, there were some exceptions in this experiment. Execution cycles under the area constraint that was equal to the 32 K byte COC had the largest speedup in P64 decode program.

(2) Memory size reduction

Figure 4.1 through 4.6 show the execution cycles v.s. cache proportion of CMC. By choosing appropriate proportion of cache, CMC using less memory area had a chance to achieve higher performance than COC with larger memory area. For example, the CMC execution cycles of 1K byte cache and 30K byte ordinary memory where cache proportion is 25% in internal memory area is almost the same as the one with 32K byte COC for JPEG decoder program. In this example, the hierarchical memory system, CMC with 4K byte cache area achieved the performance of 32K byte COC. The performance of 16K cache area CMC that had 8K byte cache and 80K byte internal memory was almost equal to the performance of 64K byte COC.

(3) Decision of cache proportion

S_{opt} depends not only on the hardware cost constraint but also on the feature of target application. In order to realize optimal execution cycles under given constraint, the decision of the most suitable cache proportion for target application becomes indispensable step in design.

The reason behind this phenomenon can be explained as follows:

(a) Case1: Cache size is larger than the size of working data set.

Internal memory reduces the access count to external memory. The effect of internal memory depends on the size of working data set of target application. When cache size is much larger than the working data set, ordinary memory has less advantage. In this case, whole working data set can be stored in the cache and the access to the ordinary memory becomes overhead. In this experiment, JPEG, MPEG and P64 required 34K byte, 404K byte and 386K byte for working data set, respectively. 128K byte cache area constraint of JPEG and 1M byte cache area constraint of MPEG and P64 are the case.

(b) Case2: Cache size is smaller than the size of working data set.

When cache proportion is much smaller, the cache miss-hit penalty will

become dominant compared to the miss-hit penalty of ordinary memory. The execution cycles also become larger in this case. 4K byte cache area constraint of JPEG and 8K byte cache area constraint of other programs are the case.

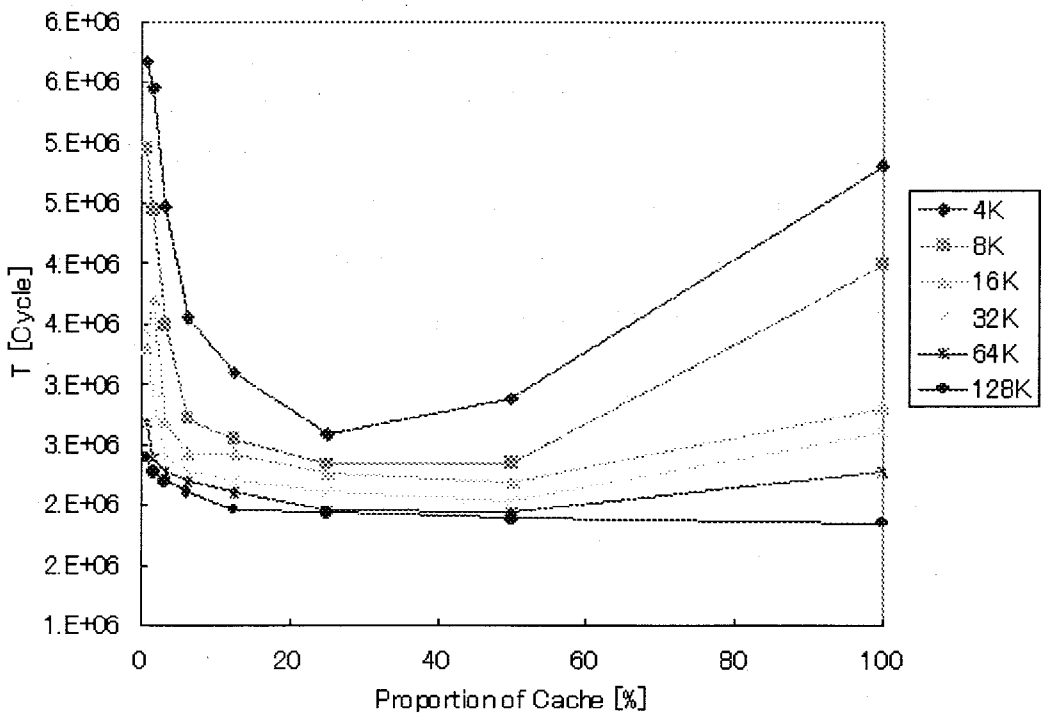


Figure 4.1: Performance v.s. Cache Proportion for JPEG Decode (128×128 pixels)

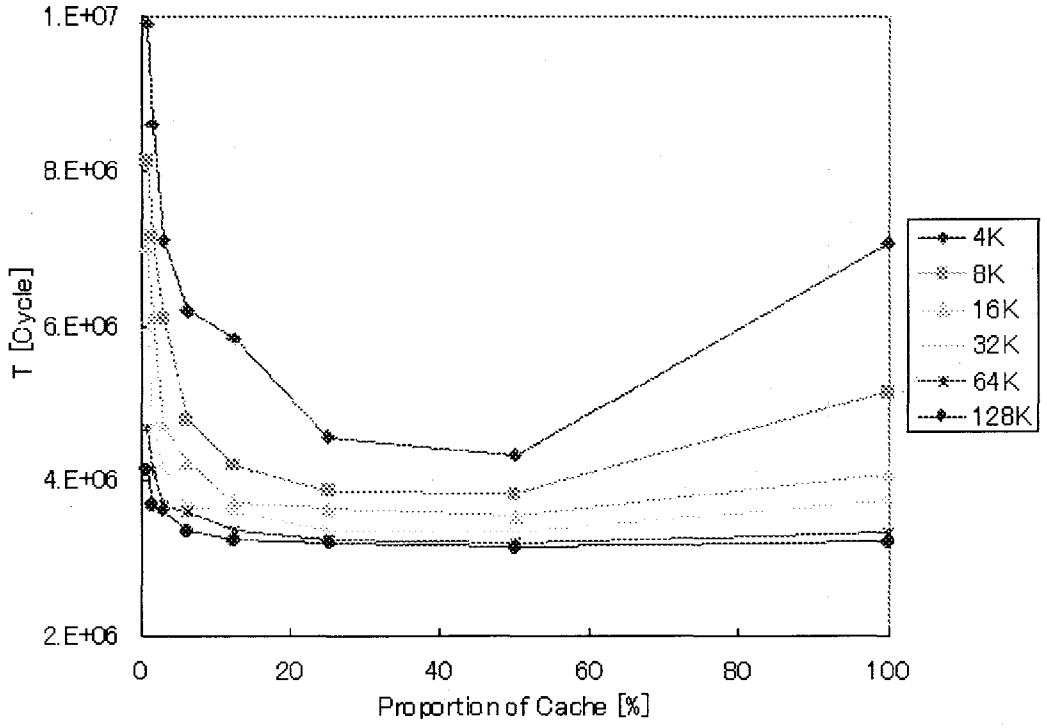


Figure 4.2: Performance v.s. Cache Proportion for JPEG Encode (128×128 pixels)

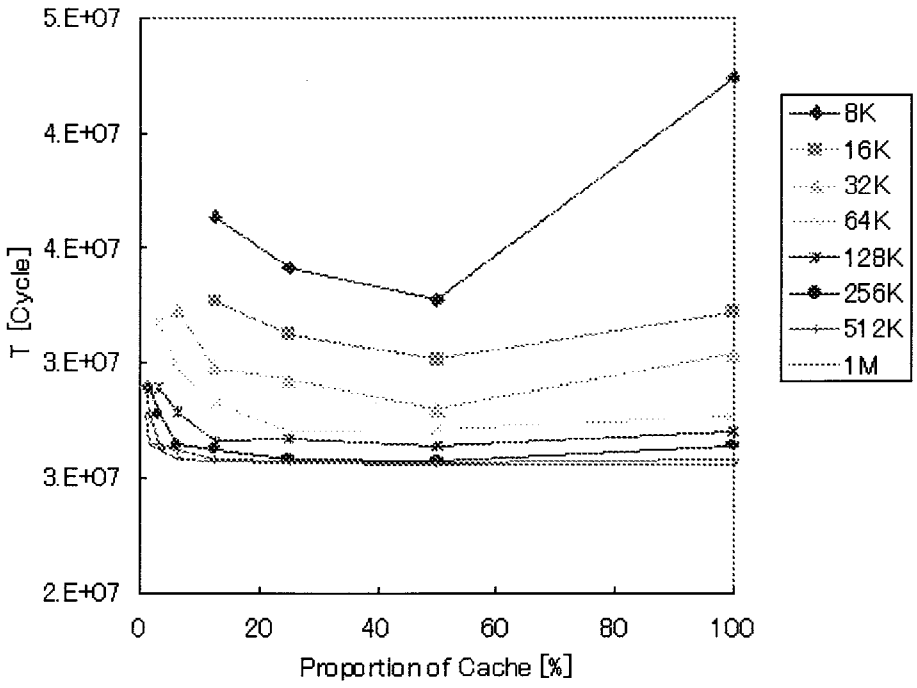


Figure 4.3: Performance v.s. Cache Proportion for MPEG Decode (4 frames)

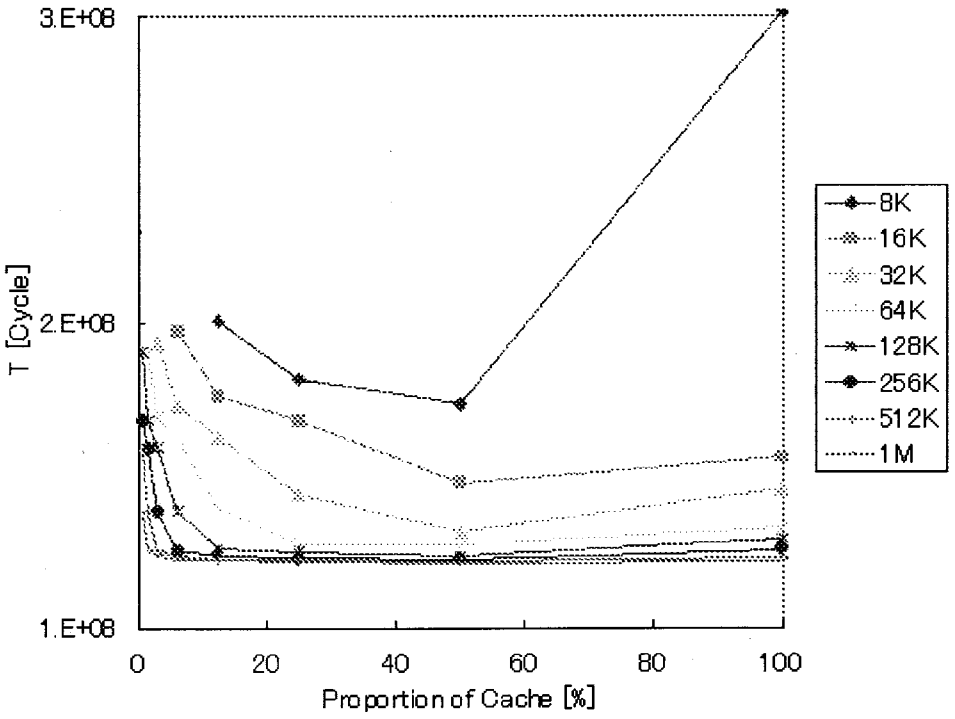


Figure 4.4: Performance v.s. Cache Proportion for MPEG Encode (4 frames)

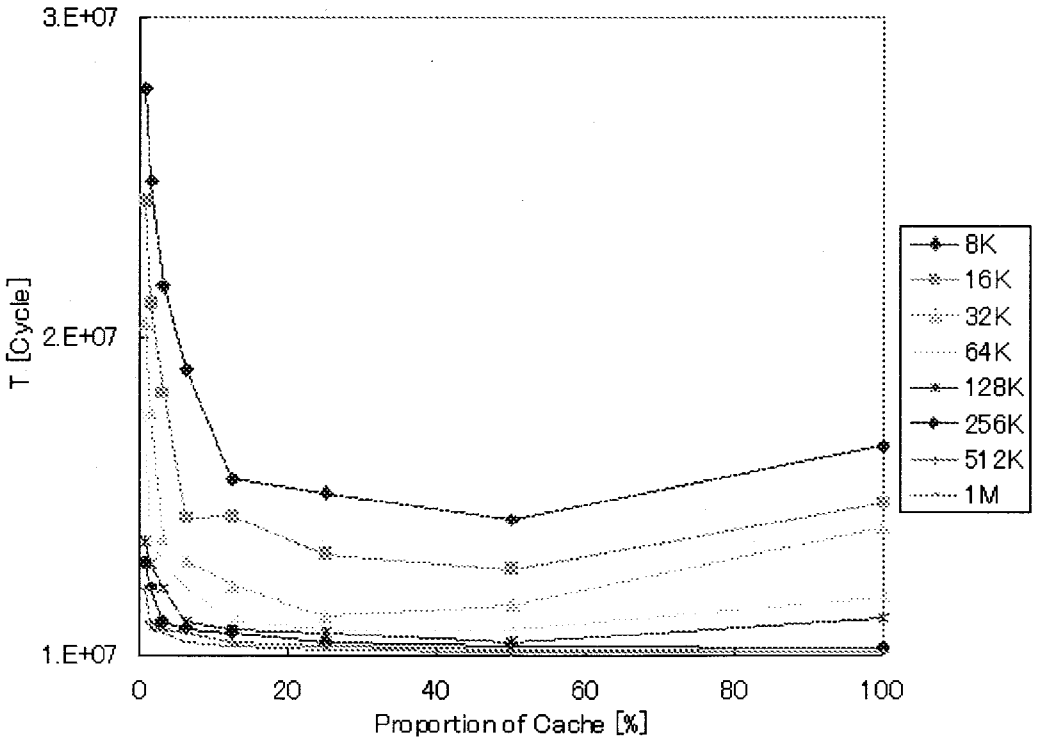


Figure 4.5: Performance v.s. Cache Proportion for P64 Decode (2 frames)

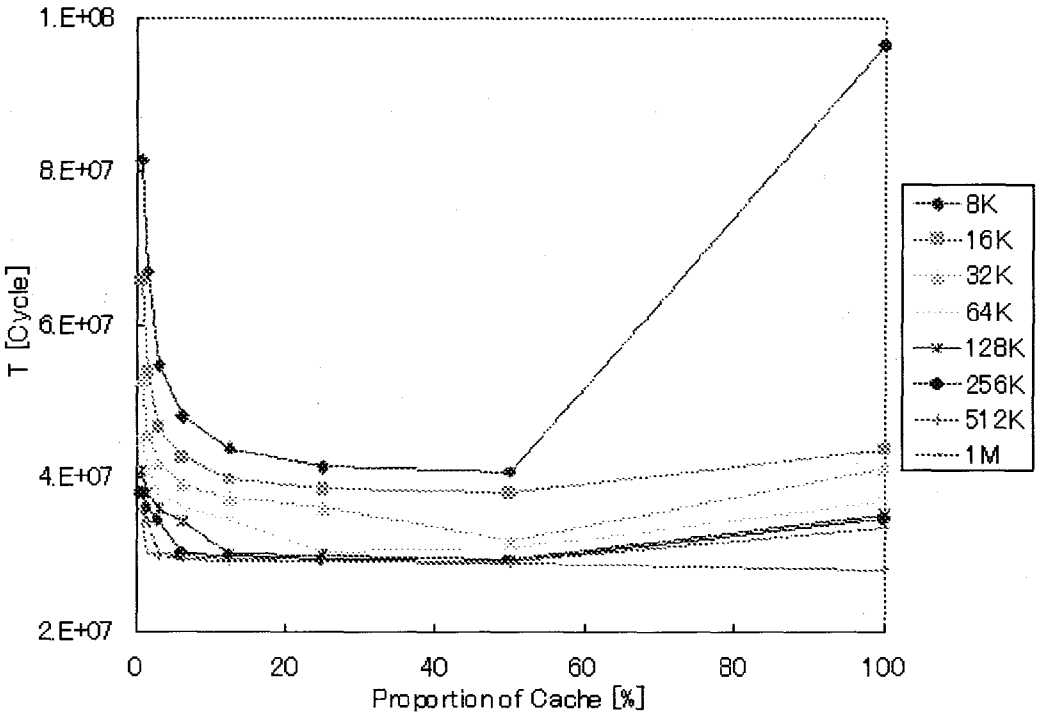


Figure 4.6: Performance v.s. Cache Proportion for P64 Encode (2 frames)

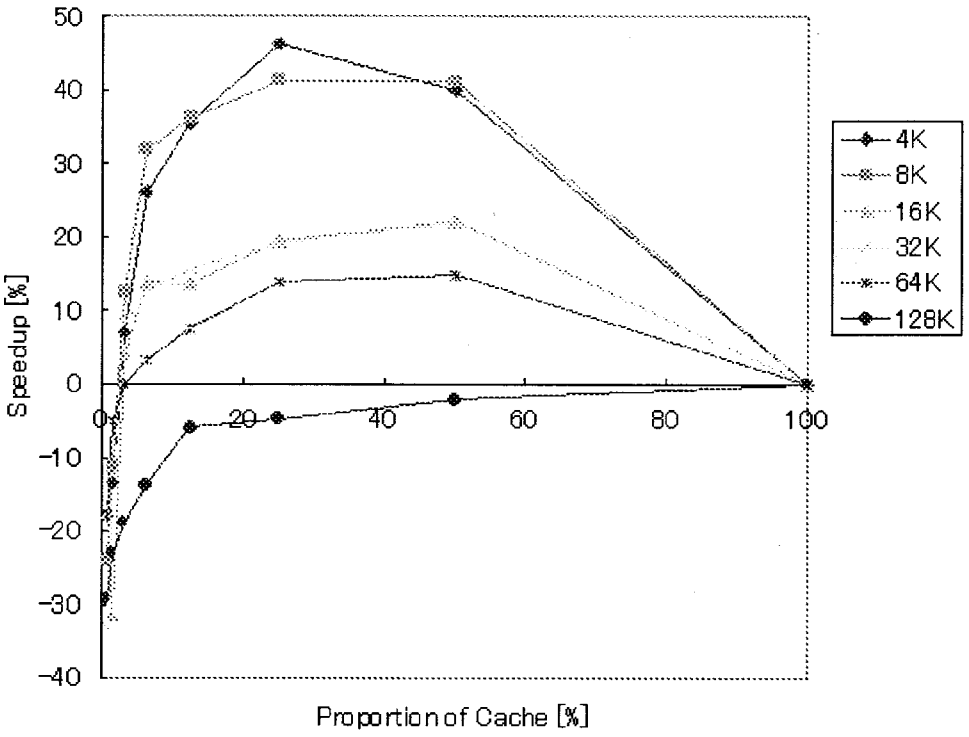


Figure 4.7: Performance Improvement for JPEG Decode (128 × 128 pixels)

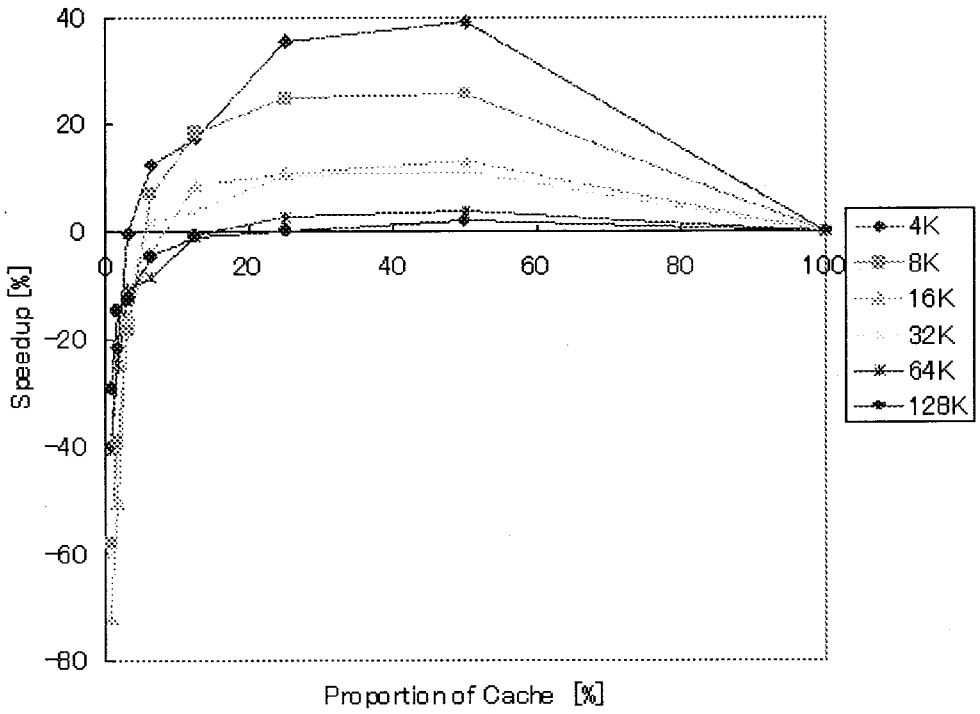


Figure 4.8: Performance Improvement for JPEG Encode (128 × 128 pixels)

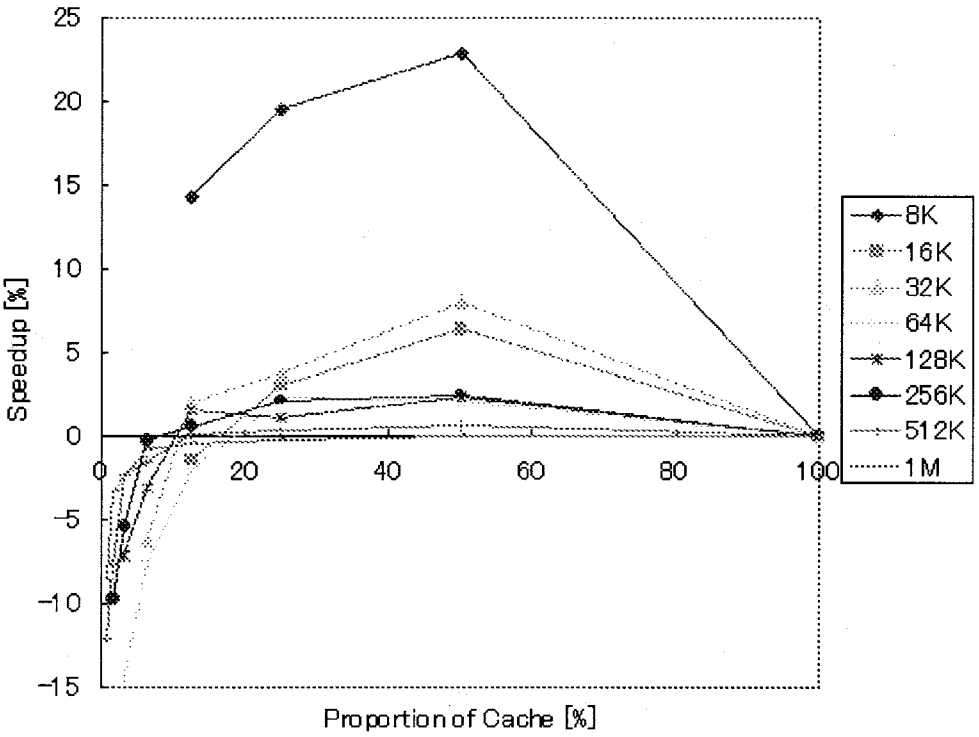


Figure 4.9: Performance Improvement for MPEG Decode (4 frames)

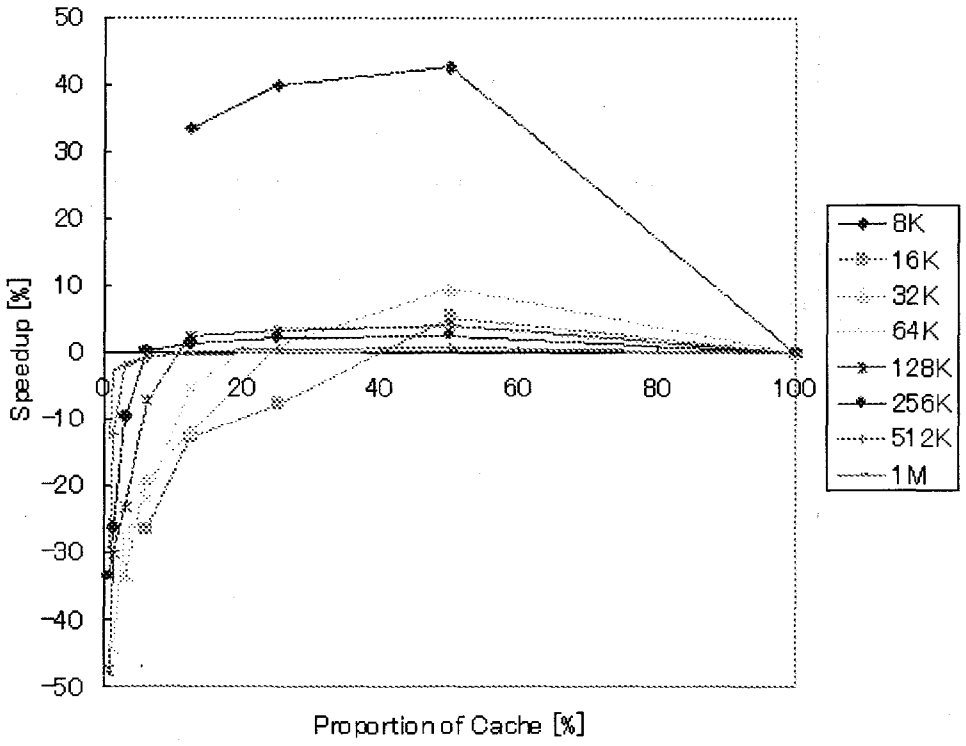


Figure 4.10: Performance Improvement for MPEG Encode (4 frames)

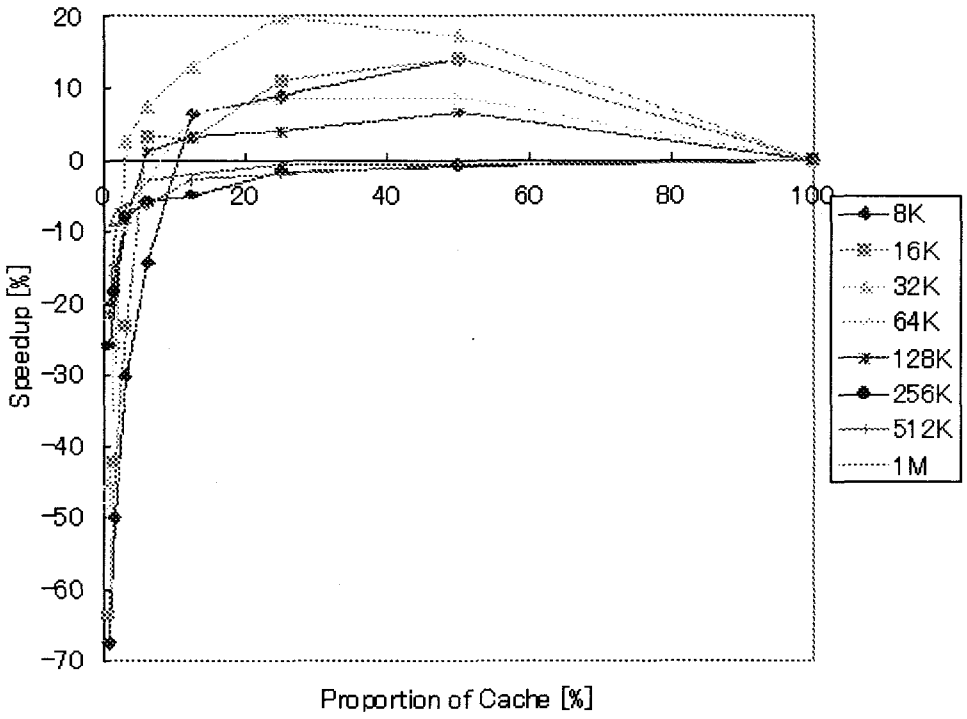


Figure 4.11: Performance Improvement for P64 Decode (2 frames)

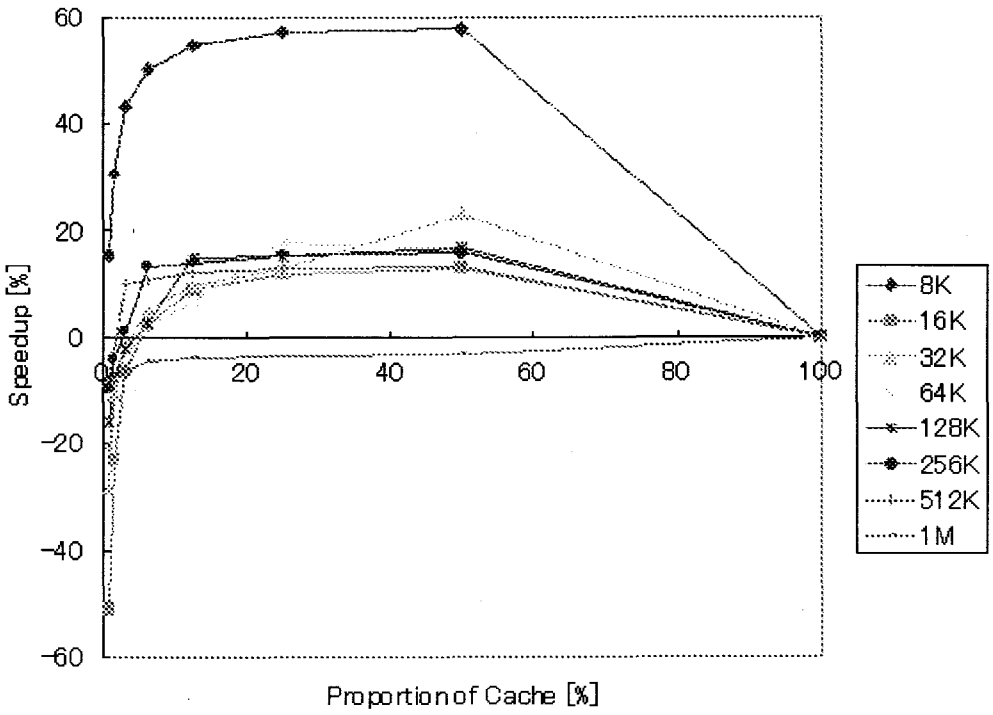


Figure 4.12: Performance Improvement for P64 Encode (2 frames)

4.4 On-chip Two level Cache Memory System

4.4.1 Characteristics of On-chip Two Level Cache Memory System

The structure of on-chip two level cache memory system is shown in Fig. 4.13. In this memory system, it is assumed that all blocks in the primary cache memory are contained in the secondary cache, and all blocks in the secondary cache memory are contained in the external memory. Integration of two level cache with a CPU core and peripherals on the same chip have following advantages.

- On-chip secondary cache memory is not necessarily a fast memory likes SRAM. Even if it is the same device as the off-chip memory like DRAM, on-chip wide bus-width enable data transfer to send/receive rapidly and the access time to the on-chip DRAM is much shorter than that to the external memory.
- The performance can be optimized by tuning the proportion of on-chip primary and secondary caches by using a profile of given set of application programs under the given hardware cost constraint of on-chip memory area.

4.4.2 Qualitative Analysis of an On-chip Two Level Cache Memory System

Let us consider the behavior of the on-chip two level cache memory system. In general, the larger the proportion of primary cache memory is in the fixed on-chip memory area, the higher hit-ratio to primary cache memory can be expected and this may lead to the reduction of the average memory accesses time. However, if the proportion of primary cache memory is too large in the fixed space, total on-chip memory size (the number of blocks) becomes smaller due to the difference of the unit costs between primary and secondary cache memories, then the access to the low-speed off-chip external memory will increase unexpectedly. Hence, the optimal proportion of two cache memories should be determined carefully by considering such characteristics.

Because the behavior of cache memory is difficult to analyze, most of the conventional optimization methods are based on simulation of memory system[85][86]. If

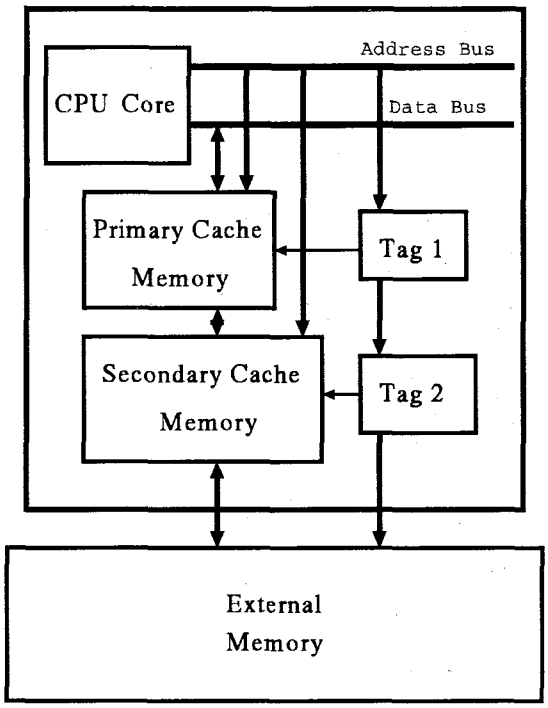


Figure 4.13: On-chip Two Level Cache Memory System.

simulation is iterated many times until the minimum point of the total access time is found, we could obtain the optimal configuration of on-chip memory. However, this takes enormous time, and are not time-efficient. Therefore, more efficient method to solve the optimal configuration is required.

4.5 Hardware Cost and Performance Model

In this section, we describe the hardware cost and performance model of on-chip two level cache memory system. We assume write back policy for both of primary and secondary cache.

4.5.1 Hardware Model

In the rest of discussion, hardware cost is represented as gate count. Fig. 4.14 shows assumed block diagram of cache memory model.

Table 4.1 shows parameters of cache configuration. The hardware cost of logic

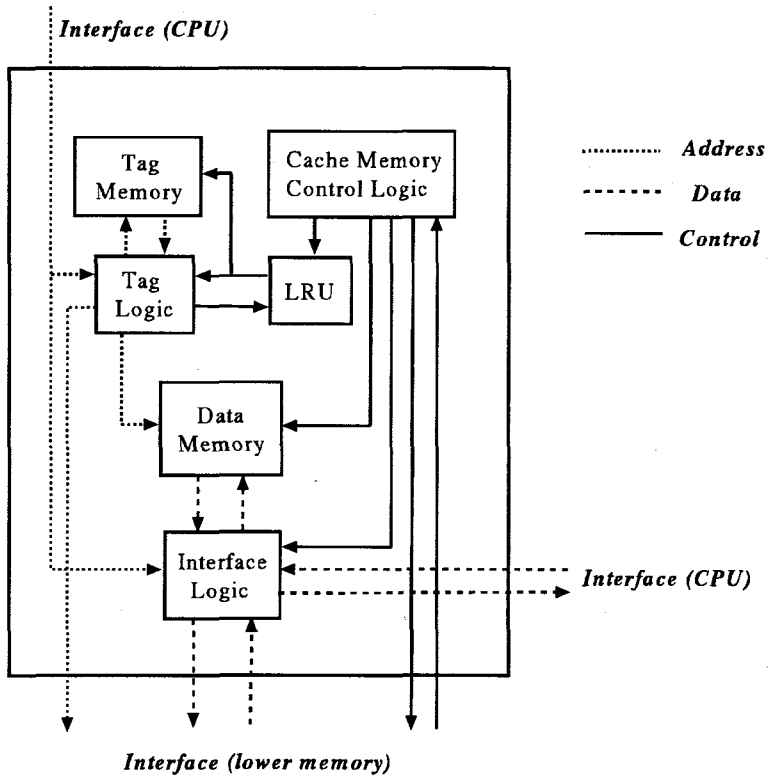


Figure 4.14: Block diagram of cache memory model.

part depends on the block size, block count and associativity of the parameters shown in table 4.1. Therefore, we represent the hardware cost of the primary and secondary cache as $Lcost_1(BS_1, BC_1, WAY_1)$, $Lcost_2(BS_2, BC_2, WAY_2)$, respectively.

The hardware cost of memory part, $Mcost$, is the sum of the tag memory cost ($TMcost$) and the data memory cost ($DMcost$). The tag memory cost can be represented as follows.

$$TMcost(tm, BS, BC) = tm \times BC \times Entry \tag{4.6}$$

$Entry$ denotes the bit count per entry of tag memory. Each entry includes dirty bit, valid bit and tag address.

$$Entry = 2 + adr\_bit - log_2(\frac{BS \times BC}{WAY}), \tag{4.7}$$

Table 4.1: Cache memory parameters.

BS	Block Size (byte)
BC	Block Count
WAY	Associativity
tm	HW Cost per Bit of Tag Memory (gate)
dm	HW Cost per Bit of Data Memory (gate)

where adr_bit represents bit width of address.

The hardware cost of data memory can be calculated as follows.

$$DMcost(dm, BS, BC) = 8 \times dm \times BS \times BC \tag{4.8}$$

From the above equations, the hardware cost of on-chip two level cache memory can be represented as follows.

$$\begin{aligned} HWcost = & \; L1\_Cost(dm_1, tm_1, BS_1, BC_1, WAY_1) \\ & + L2\_Cost(dm_2, tm_2, BS_2, BC_2, WAY_2) \end{aligned} \tag{4.9}$$

$$\begin{aligned} L1\_Cost(dm_1, tm_1, BS_1, BC_1, WAY_1) &= Lcost_1(BS_1, BC_1, WAY_1) \\ &+ Mcost(dm_1, tm_1, BS_1, BC_1) \end{aligned} \tag{4.10}$$

$$\begin{aligned} L2\_Cost(dm_2, tm_2, BS_2, BC_2, WAY_2) &= Lcost_2(BS_2, BC_2, WAY_2) \\ &+ Mcost(dm_2, tm_2, BS_2, BC_2) \end{aligned} \tag{4.11}$$

4.5.2 Performance Model

When C and T denote total memory access count and total memory access time respectively, C and T can be represented as follows.

$$C = C_1 \tag{4.12}$$

$$\begin{aligned} T &= T_1 \times C_1 + T_2 \times C_2 + T_{ex} \times C_{ex} \\ &+ DT_1 \times DC_1 + DT_2 \times DC_2, \end{aligned} \tag{4.13}$$

where C_1, C_2 and C_{ex} are access counts to primary cache, secondary cache and main memory, respectively. T_1, T_2 and T_{ex} are access time to primary cache, secondary cache and main memory, respectively. DT_1 and DT_2 are overhead time required to write the block data to be replaced out of the cache should be written back to the lower memory. DC_1 and DC_2 is write-back counts on primary and secondary cache.

4.6 Performance Optimization Method

4.6.1 Estimation of Average Memory Access Time

First, when H_1 and H_2 denote hit-ratio to primary and secondary cache, average memory access time can be represented as follows:

$$\begin{aligned} \overline{T} &= T_1 + T_2 \times (1 - H_1) + T_{ex} \times (1 - H_1)(1 - H_2) \\ &+ DT_1 \times D_1 + DT_2 \times D_2, \end{aligned} \tag{4.14}$$

where H_2 means local hit-ratio. D_1 and D_2 denote probability of write-back on primary and secondary cache, respectively.

Secondly, the condition for hardware cost constraint can be represented as follows with models in Section 4.5.1.

$$\begin{aligned} & L1\_Cost(dm_1, tm_1, BS_1, BC_1, SET_1) \\ + \; & L2\_Cost(dm_2, tm_2, BS_2, BC_2, SET_2) \leq Cost \end{aligned} \tag{4.15}$$

where $Cost$ means hardware cost constraint(gate count).

Next, suppose that a cache memory size, block size and a memory access sequence are given, the hit-ratio to the cache memory can be uniquely determined if other parameters of the cache memory, such as write policy, block size, associativity, and replacement policy, are fixed. Then, we suppose that a function $hit(blockcount)$ returns the hit-ratio to the cache memory possessing the given block counts under fixed parameters. Using this function, hit-ratio of primary cache memory, H_1, is represented as

$$H_1 = hit(BC_1). \tag{4.16}$$

When H_{12} denotes the hit-ratio of on-chip memory, H_{12} can be represented as follows.

$$H_{12} = H_1 + (1 - H_1) \times H_2 \tag{4.17}$$

Since all the data in the primary cache is contained in the secondary cache, H_{12} is equal to the hit-ratio of cache memory whose block count is BC_2. Therefore,

$$H_{12} = hit(BC_2). \tag{4.18}$$

From Eq. (4.17) and (4.18), we have

$$(1 - H_1) \times H_2 = hit(BC_2) - hit(BC_1). \tag{4.19}$$

As D_1 and D_2 is determined by primary and secondary cache size, we assume $dirty()$ as follows.

$$D_1 = dirty(BC_1) \tag{4.20}$$

$$D_2 = dirty(BC_2) \tag{4.21}$$

From Eq. (4.16), (4.19), (4.21) and (4.21), Eq. (4.14) can be represented as follows.

$$\begin{aligned} \overline{T} &= T_1 + T_2 \times (1 - H_1) + T_{ex} \times (1 - H_1)(1 - H_2) \\ &+ DT_1 \times dirty(BC_1) + DT_2 \times dirty(BC_2) \\ &= T_1 + (T_2 + T_{ex}) \times (1 - H_1) - T_{ex} \times (1 - H_1) H_2 \\ &+ DT_1 \times dirty(BC_1) + DT_2 \times dirty(BC_2) \end{aligned}$$

$$
\begin{aligned}
&= T_1 + (T_2 + T_{ex}) \times (1 - hit(BC_1)) - T_{ex} \times (hit(BC_2) - hit(BC_1)) \\
&+ DT_1 \times dirty(BC_1) + DT_2 \times dirty(BC_2) \\
&= T_1 + T_2 \times (1 - hit(BC_1)) + T_{ex} \times (1 - hit(BC_2)) \\
&+ DT_1 \times dirty(BC_1) + DT_2 \times dirty(BC_2) \qquad (4.22)
\end{aligned}
$$

4.6.2 Algorithm

From Eqs. (4.15) and (4.22), an optimal sizes of primary and secondary cache can be found by searching the combination of two cache memory sizes to minimize the average access time($\overline{T}$ in Eq. (4.22)), which satisfies the cost constraint Eq. (4.15). The algorithm to find the optimal sizes are outlined in Fig. 4.15.

4.6.3 Hit-ratio Prediction

The remaining problem to calculate $\overline{T}$ in Eq. (4.22) is how to determine function $hit()$. To caluculate $hit()$ in Fig. 4.15 with cache simulator spends enormous time. In this subsection, efficient implementation of function $hit()$ is described.

In the following experiments, following assumptions are made.

1. Replacement policy is LRU (Least Recently Used).
2. Association method is fully associative.

When main memory consists of M blocks, the hit-ratio of cache memory can be estimated by the following expression:

$$
\text{hit-ratio} = \sum_{i=0}^{M-1} p_i \times q_i \qquad (4.23)
$$

where p_i : probability that block i is accessed

q_i : probability that block i exists in the cache memory

Let AC_i denote the access count to block i, then p_i can be represented as follows:

```
INPUT:      HW cost constraint (Cost)
            Block Size (BS1, BS2)
            Associativity (WAY1, WAY2)
            Access Time (T1, T2, Tex)
            Write-back Overhead Time (DT1, DT2)
            Memory Cell Size (dm1, dm2, tm1, tm2)
OUTPUT:     Optimal Cache Size (Sopt1, Sopt2)
            Minimum Average Memory Access Time (T̄min)
Algorithm:
    begin
      BC1 = 0;
      BC2 = L2BC(Cost);
      T̄min = T2 + Tex * (1 − hit(BC2)) + DT2*dirty(BC2);
      Sopt1 = 0;
      Sopt2 = BC2 * BS2;
      BC1 = 1;
      BC2 = L2BC(Cost-L1_Cost(dm1, tm1, BS1, BC1, WAY1);
      while BC1 < BC2 do
        begin
          T̄ = T1 + (T2 + Tex) * (1 − hit(BC1))
                 + Tex * (hit(BC2) − hit(BC1))
                 + DT1*dirty(BC1) + DT2*dirty(BC2);
          if (T̄ < T̄min) then
            begin
              T̄min = T̄;
              Sopt1 = BC1 * BS1;
              Sopt2 = BC2 * BS2
            end
          BC1++;
          BC2 = L2BC(Cost-L1_Cost(dm1, tm1, BS1, BC1, WAY1);
        end
      BC1 = L1BC(Cost);
      BC2 = 0;
      T̄min = T1 + Tex * (1 − hit(BC1)) + DT2*dirty(BC1);
      if (T̄ < T̄min) then
        begin
          T̄min = T̄;
          Sopt1 = BC1 * BS1;
          Sopt2 = 0
        end
    end.
    function L1BC(Budget)
      begin
        BC = 0;
          whileL1_Cost(dm1, tm1, BS1, BC1, WAY1) ≤ Budget do
BC++;
        return(BC)
      end;
    function L2BC(Budget)
      begin
        BC = 0;
          whileL2_Cost(dm2, tm2, BS2, BC2, WAY2) ≤ Budget do
BC++;
        return(BC)
      end;
```

Figure 4.15: Algorithm to solve the optimal cache memory sizes.

$$p_i = AC_i/C \tag{4.24}$$

where C denotes total access count, i.e. $C = \sum_{i=0}^{M-1} AC_i$. Note that p_i is independent of memory size and other cache parameters, but q_i is not. q_i can be determined by the following procedure. Let $BA_{i,k}$ denote the kth access to block$i$.

Step1. For each $BA_{i,k}$ (i=0,..,M, k=1,..,$AC_i - 1$), investigate the *distance*s which mean the number of different blocks been accessed between $BA_{i,k}$ and $BA_{i,k+1}$. Let $DB_{i,k}$ denote the *distance* of $BA_{i,k}$.

Step2. For each i (i=0,..,M), count the number of $DB_{i,k}$'s (k=1,..,$AC_i - 1$) which are equal to j, and represent the number as $AC_i(j)$.

Step3. Calculate q_i's (i=0,..,M) by dividing the sum of $AC_i(j)$'s by AC_i where j is less than the number of blocks in the cache memory, denoted as N.

Figure 4.16: Procedure to calculate q_i.

The idea behind this procedure in Fig. 4.16. is as follows. The block stored in the cache memory by a certain access becomes the least-recently-used block after succeeding accesses to $N - 1$ difference blocks. Therefore, the block is replaced out by the access that causes the next replacement of the memory. Hence, when the cache memory consists of N blocks, q_i can be determined by the following equation:

$$q_i = \frac{\sum_{j=0}^{N-1} AC_i(j)}{AC_i} \tag{4.25}$$

4.6.4 Example

We show an illustrative example to calculate $hit()$. Let us determine the function $hit()$ for the access sequence in the following order. The numbers represent accessed blocks in the main memory (N=10, C=23).

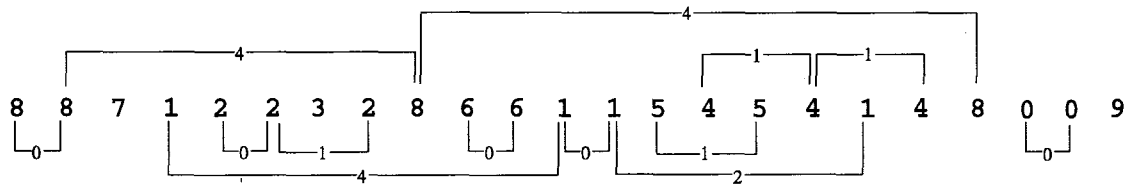
8 8 7 1 2 2 3 2 8 6 6 1 1 5 4 5 4 1 4 8 0 0 9

First, AC_i and p_i are found from the trace data as follows.

i	0	1	2	3	4	5	6	7	8	9
BC_i	2	4	3	1	3	2	2	1	4	1
p_i	2/23	4/23	3/23	1/23	3/23	2/23	2/23	1/23	4/23	1/23

Next, q_i can be obtained as follows.

Step1. The *distances* between $BA_{i,k}$ and $BA_{i,k+1}$(i=0,..,9) are shown in the next figure. The small numbers represent the *distances*.



From the above *distances*, $DB_{i,k}$'s are calculated as shown in the following table:

Access *Distances* of $BA_{i,k}$.

$DB_{i,k}$		block No. i									
		0	1	2	3	4	5	6	7	8	9
k	1	0	4	0	-	1	1	0	-	0	-
	2	-	0	1	-	1	-	-	-	4	-
	3	-	2	-	-	-	-	-	-	4	-

Step2. From the result of Step 1, $AC_i(j)$, access *distances* of block i, are calculated as shown in the following table:

Access count table of block i.

$AC_i(j)$		block No. i									
		0	1	2	3	4	5	6	7	8	9
j	0	1	1	1	0	0	0	1	0	1	0
	1	0	0	1	0	2	1	0	0	0	0
	2	0	1	0	0	0	0	0	0	0	0
	3	0	0	0	0	0	0	0	0	0	0
	4	0	1	0	0	0	0	0	0	2	0

Step3. q_i's (i=0,..,9) are calculated by Eq. (4.25) and the above table. The results are shown as follows:

Probability that block i exists in the memory.

q_i		block No. i									
		0	1	2	3	4	5	6	7	8	9
N	0	0	0	0	0	0	0	0	0	0	0
	1	1/2	1/4	1/3	0	0	0	1/2	0	1/4	0
	2	1/2	1/4	2/3	0	2/3	1/2	1/2	0	1/4	0
	3	1/2	2/4	2/3	0	2/3	1/2	1/2	0	1/4	0
	4	1/2	2/4	2/3	0	2/3	1/2	1/2	0	1/4	0
	5	1/2	3/4	2/3	0	2/3	1/2	1/2	0	3/4	0

From the tables of p_i and q_i, and Eq. (4.23), the relation between the cache memory size (number of block:M) and the hit-ratio is calculated as follows:

M	0	1	2	3	4	5	$\cdots$	10
$hit()$	0	5/21	9/21	10/21	10/21	13/21	$\cdots$	13/21

The advantage of this method is that once $AC_i(j)$ is investigated on all blocks, q_i can be calculated by only adding $AC_i(j)$. The proposed method can avoid the re-investigation of $AC_i(j)$, even if the cache memory size is changed. Note that all values $BA_{i,k}$, $AC_i(j)$, AC_i are given from profiling the application program by an *analyzer* to be addressed in the next section.

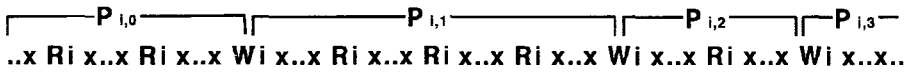
4.6.5 Write-Back Prediction

In a cache memory with write-back policy, when the replacement of dirty block (called write-back) occurred, the block should be write-back to the lower memory. In this subsection, the prediction method of write-back count is described to consider the overhead time caused by the write-back.

The accesses to block i can be divided into read accesses and write accesses as follows, where **R** and **W** represent read access and write access, respectively, and **x** represent the access to any block except block i.

..x Ri x..x Ri x..x Wi x..x Ri x..x Ri x..x Ri x..x Wi x..x Ri x..x Wi x..x..

Only when the write access are done, the block becomes dirty. Here, we divide the accesses to block i into w_i periods, and represent kth period as $P_{i,k}$ as follows. w_i represents the write access count to block i.



Next, we focus on period $P_{i,k}$. When the block count of the cache memory is N, the dirty block in the period $P_{i,k}$ is not replaced from cache memory, unless the access distances of the succeeding accesses to block i exceed $N-1$. Therefore, all of the access distances in the period $P_{i,k}$ is less than N, write-back does not occur. Also, once a write-back occurs, another write-back does not occur in this period, because the block i is not dirty until the next write access to block i. Hence, when the maximum distance in the period $P_{i,k}$ is $PD_{i,k}$, the following relations hold.

(1) if $PD_{i,k} \leq N-1$ then write-back doesn't occur,

(2) if $PD_{i,k} > N-1$ then one write-back occur.

When $PC_i(j)$ is the number of period $P_{i,k}$'s $(k = 0, .., w_i)$ whose $PD_{i,k} = j$, the write-back count to block i is as follows.

$$\sum_{j=N} PC_i(j) = w_i - \sum_{j=0}^{N-1} PC_i(j) \tag{4.26}$$

Therefore, when the main memory has M blocks, from Eq. 4.26, the write-back count is as follows.

$$\sum_{i=0}^{M-1} w_i - \sum_{i=0}^{M-1}\sum_{j=0}^{N-1} PC_i(j)$$
$$= WC - \sum_{i=0}^{M-1}\sum_{j=0}^{N-1} PC_i(j), \tag{4.27}$$

where WC represents total count of write accesses.

4.7 Experiments

4.7.1 Assumptions

The following experiments were performed on the WS $NEWS50000$(200MHz). Table 4.2 shows parameter values of on-chip memory system.

Table 4.2: Parameter values.

adr_bit	bit width of address	32 bit
$word\_byte$	bytes per word	4 byte
BS_1	block size of primary cache	32 byte
BS_2	block size of secondary cache	32 byte
tm_1	cost per tag memory bit of primary cache	0.31 gate
tm_2	cost per tag memory bit of secondary cache	0.31 gate
dm_1	cost per data memory bit of primary cache	0.31gate
dm_2	cost per data memory bit of secondary cache	0.031 gate
T_1	access time of primary cache	2 cycle
T_2	access time of secondary cache	3 cycle
T_{ex}	access time of main memory	32 cycle
DT_1	write-back overhead time of primary cache	4 cycle
DT_2	write-back overhead time of secondary cache	33 cycle

Some values of table 4.2 were determined based on the VHDL simulation of memory system model description. To solve the hardware cost of cache memory, this description was synthesized by a logic synthesis tool. The hardware cost can be estimated using following formula.

$$
\begin{aligned}
Lcost_1(BS_1, BC_1, WAY_1) &= 2.9 \times \frac{BC_1}{WAY_1} - 4.2 \times log_2 \frac{BC_1 \times BS_1}{WAY_1} \\
&+ 261.2 \times WAY_1 - 7.2 \times WAY_1 \times log_2 \frac{BC_1 \times BS_1}{WAY_1} \\
&+ 8.1 \times BC_1 \times WAY_1 - 5.3 \times BC_1 + 5716.5 \quad (4.28) \\
Lcost_2(BS_2, BC_2, WAY_2) &= 2.9 \times \frac{BC_2}{WAY_2} - 4.2 \times log_2 \frac{BC_2 \times BS_2}{WAY_2} \\
&+ 261.2 \times WAY_2 - 7.2 \times WAY_2 \times log_2 \frac{BC_2 \times BS_2}{WAY_2} \\
&+ 8.1 \times BC_2 \times WAY_2 - 5.3 \times BC_2 + 4416.9 \quad (4.29)
\end{aligned}
$$

The hardware cost of single cache memory was larger than those cost due to its intricate interface mechanism.

$$
\begin{aligned}
Lcost(BS, BC, WAY) &= 2.9 \times \frac{BC}{WAY} - 4.2 \times log_2 \frac{BC \times BS}{WAY} \\
&+ 261.2 \times WAY - 7.2 \times WAY \times log_2 \frac{BC \times BS}{WAY} \\
&+ 8.1 \times BC \times WAY - 5.3 \times BC + 10336 \quad (4.30)
\end{aligned}
$$

4.7.2 Accuracy of Proposed Method

Table 4.3 shows the relation between hit-ratio and size of cache memory. Sample application is a JPEG decoder program. The values in column 2 are the predicted values, and values in column 3 are the results of cache simulation using *dinero* simulator. From this table, it is known that function *hit*() returns very accurate hit-ratio values.

Table 4.4 shows the relation between write-back count and cache memory size. Where sample application is JPEG decoder program. From this table, the proposed method of write-back prediction is found very accurate.

Fig. 4.17 shows the average memory access time obtained by the cache simulation and by the proposed method. Both of the primary and secondary caches were fully associative caches. The hardware cost constraint was 100 Kgate. This figure shows that the proposed method solves average memory access time very accurately against

Table 4.3: Relation between hit-ratio and size of cache memory – *hit*() v.s. simulation.

JPEG decoder program

cache size (Byte)	(a) *hit*()	(b) Simulation	Difference (b) – (a)
256	0.832393	0.832394	0.000001
512	0.932639	0.932639	0
1K	0.958007	0.958007	0
2K	0.969314	0.969315	0.000001
4K	0.997620	0.997624	0.000004
8K	0.998215	0.998215	0
16K	0.998364	0.998366	0.000002

Table 4.4: Relation between write-back count and size of cache memory – *dirty*() v.s. simulation.

JPEG decoder program

cache size (Byte)	(a) *dirty*()	(b) simulation
256	121916	121916
512	50237	50237
1K	33256	33256
2K	21689	21689
4K	3488	3488
8K	2607	2607
16K	2250	2250

fully associative cache. In this experiment, the optimal configuration obtained by the proposed method and the cache simulation are the same.

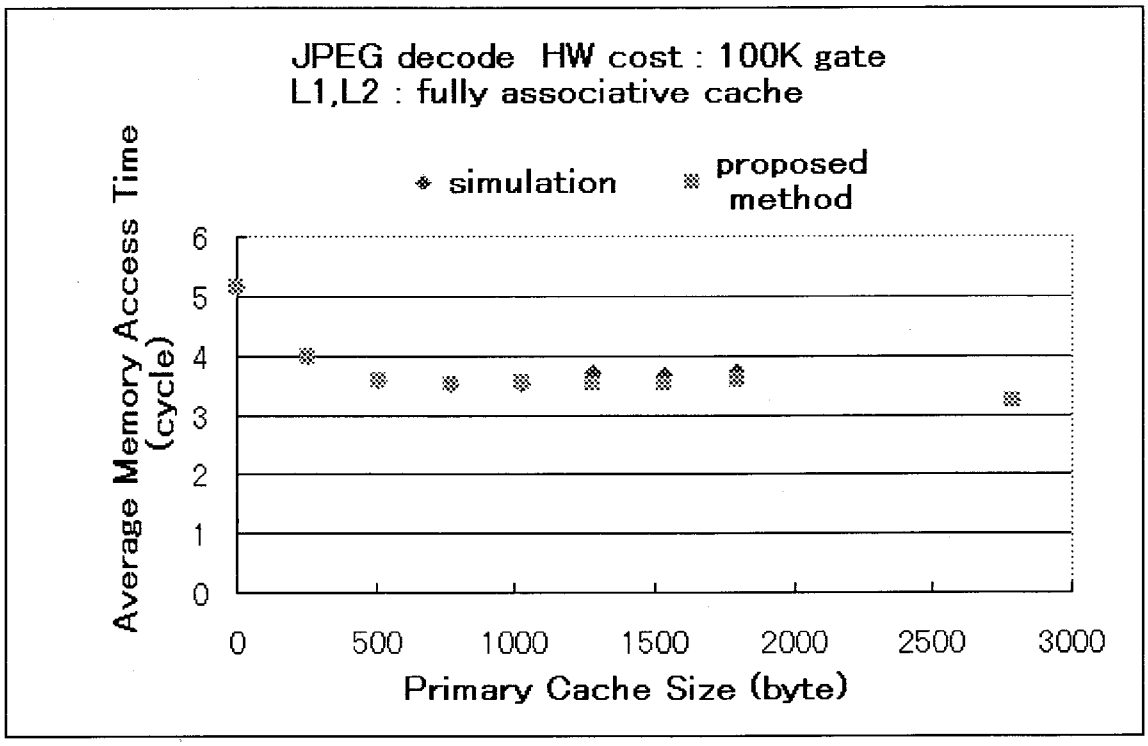


Figure 4.17: The average memory access time of fully associative cache.
HW cost constraint: 100K gate

The computation time of the proposed method and the simulation method are shown in table 4.5 and 4.6, respectively. Both of the primary and secondary cache were 2-way associative cache. The proposed method can be implemented in two phases: *analyzer* and *solver*. The *analyzer* is used to analyze trace data, and *solver* is used to find the optimal configuration based on the algorithm shown in Fig. 4.15, using the results of *analyzer*. The computation time of the proposed

method is the sum of the *analyzer* execution time and *solver* execution time. The *analyzer* does not depend on hardware cost constraint. The computation time of the simulation method is the product of the execution time of cache simulation and the iteration count. From table 4.5 and 4.6, it is known that the proposed method is much more efficient than a simulation method.

Table 4.5: The computation time of the proposed method (unit: second)

HW cost		JPEG decoder	JPEG encoder	MPEG decoder
	analyzer	16	27	398
20K gate	*solver*	1.2	1.3	2.1
	total	17.2	28.3	400.1
30K gate	*solver*	1.2	1.4	2.1
	total	17.2	28.4	400.1
50K gate	*solver*	1.3	1.5	2.2
	total	17.3	28.5	400.2

Table 4.6: The computation time of the simulation method (unit: second)

HW cost		JPEG decoder	JPEG encoder	MPEG decoder
	cache simulation	22	31	324
20K gate	iteration count	38	38	38
	total	836	1178	12312
30K gate	iteration count	78	78	78
	total	1716	2418	25272
50K gate	iteration count	158	158	158
	total	3476	4898	51192

4.7.3 Effectiveness for Non-fully Associative Cache Memory

We applied the proposed method to non-fully associative cache. Fig. 4.18 shows the result against 2-way primary cache and 4-way secondary cache. Sample program was JPEG decoder program. Although the estimation of the average memory access time

included larger error than that for fully associative cache, the difference between the average memory access time of the configuration obtained by the proposed method and by the simulation was within about 5%, which were not so large. Fig. 4.19 shows the result against JPEG encoder program.

According to the results, the accuracy of predictions is lower using proposed method for non-fully associative cache memory. However, the estimation error is not so large, then the exploration time to decide the optimal configuration of non-fully assciative cache memory would be reduced by using proposed method.

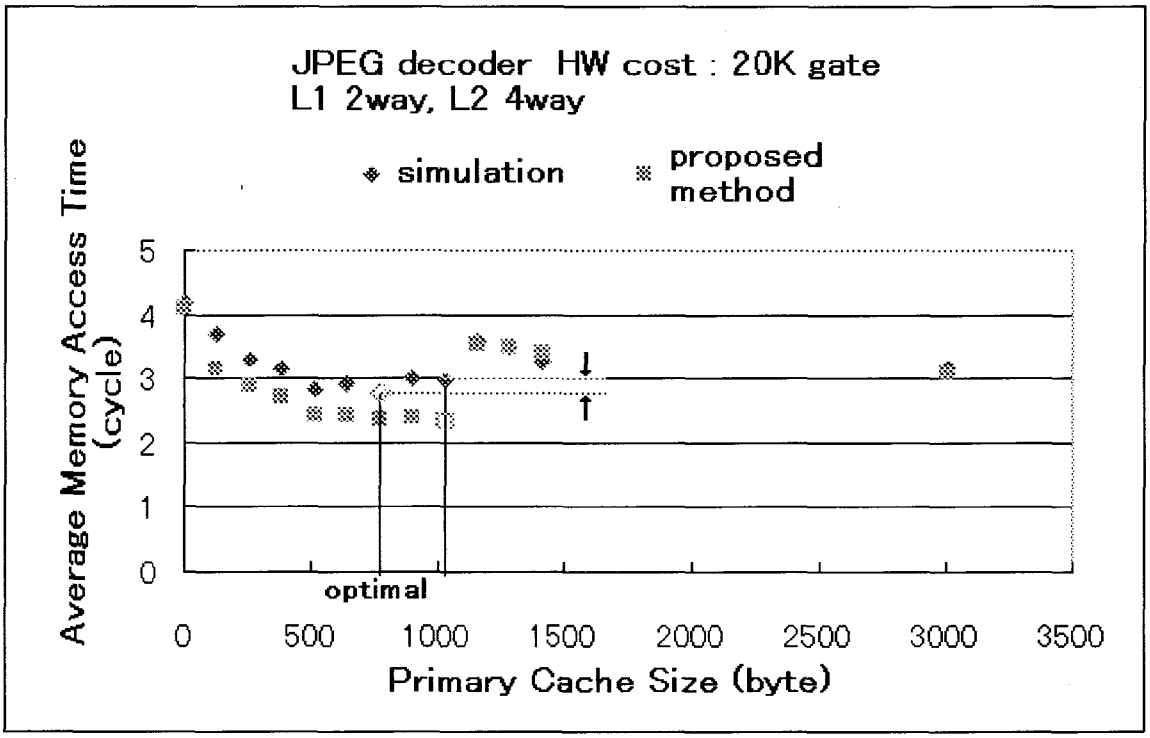


Figure 4.18: The average memory access time of non-fully associative cache.

HW cost constraint: 20K gate

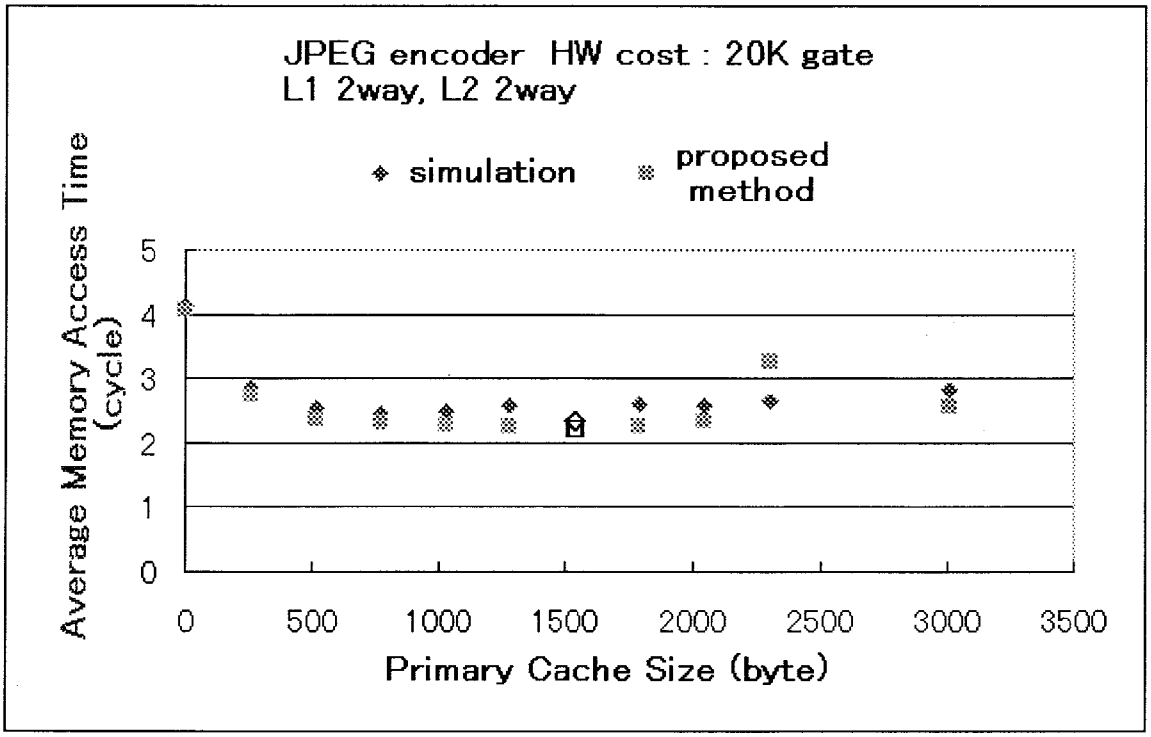


Figure 4.19: The average memory access time of non-fully associative cache.

HW cost constraint: 20K gate

Chapter 5

Conlusions and Future Work

In this thesis, a design optimization method for Application Specific Integrated Processor including dedicated CPU core and on-chip hierarchical memory system is proposed. The concluding remarks are described in sections 5.1 and 5.2. In section 5.3, future work in this research area is discussed.

5.1 CPU Core Design Method for ASIPs

In chapter 2, the architecture and ASIC implementation of FSP-3 were described for a case study of an ASIP. FSP-3 is expected to be an efficient special purpose microprocessor for flexible servomotor control systems, because FSP-3 has an appropriate architecture for this application and reduces external data accesses.

Chapter 3 describes PEAS-I: a hardware/software codesign system for ASIP development. The effectiveness and efficiency of the system were confirmed through several experiments. According to the primary experimental results, PEAS-I system gives accurate estimation of the chip area and performance of ASIPs before the detailed hardware design is completed. The experimental results also show that PEAS-I system is able to generate both hardware design and a set of application program development tools for a typical size ASIP within several hours.

In a general-purpose CPU, the tuning of the architecture to a specific application by changing hardware resources and instruction set architecture is very difficult. PEAS-I system can decide the best instruction set architecture and its implementation method from analytical results of the application programs under the given

design constraints. Thus, PEAS-I system can reduce the execution time of application programs. Moreover, the functionality of ASIP can be improved by considering the tradeoffs of the amount of hardware of the CPU core, memory, peripheral circuits, etc.

5.2 Memory Optimization Method for ASIP

In chapter 4, a performance optimization method is proposed for hierarchical memory system for ASIPs, which consists of on-chip fast cache memory, a large amount of on-chip ordinary memory and a huge off-chip memory. Using the hierarchical on-chip memory system, the performance of an ASIP could be improved up to 50% compared to that with conventional cache memory system.

The performance optimization method includes hit-ratio prediction, write-back penalty prediction, and average memory access time estimation. From the experimental results, it is known that the proposed method can decide an optimal configuration much more efficiently than conventional optimization methods based on the iteration of cache simulation. The proposed method can estimate the average memory access time very accurately for fully associative caches. Even when the cache memory is non-fully associative, the performance difference between the configurations obtained by the proposed method and by the cache simulation was about 5%.

5.3 Future Work

The proposed method based on Hardware/Software codesign approach will be able to reduce the design time and effort, and improve the design quality of ASIPs. This approach has following benefits:

(1) higher abstraction level of design,

(2) accurate estimation of design quality, and

(3) optimization design assistance.

However, in order to design ASIPs using deep submicron technology and to improve the performance of ASIPs, following problems should be addressed.

5.3.1 Target CPU Core Architecture

Currently, the processor model to be generated by PEAS-I system has a simple RISC type pipeline architecture. When an embedded system is required much more high performance, other processor architectures such as VLIW shuold be considered for CPU core of ASIP, such as for multimedia application domain. Then, new project PEAS-II for VLIW processor based ASIP development has been started.

5.3.2 Specific Hardware Design Methodology

One of the major components of ASIP is a specific hardware such as DSP and co-processor. PEAS-I system has a concept of "Extended Functionarities" that can be realize by a specific hardware. However, PEAS-I system has no function to support such hardware modules. Therefore, the protocol and interface to communicate between CPU and specific hardware is needed. Also the scheme to use a "Intellectual Property (IP)" in a ASIP is another future work.

5.3.3 Flexible Hardware Model

In the deep submicron technology, wiring delay will become dominant rather than the switching delay of a gate, and wiring capacitance will become dominant in power consumption. While simple conventional circuit models were effective in the design method for 0.5μm technology or over, more sophisticated models should be used in deep submicron technology. Where, the resistance, mutual capacitance and mutual inductance of wiring cannot be ignored as well as parasitic capacitance.

Because the new wiring model takes into account the features of deep submicron technology, the estimation is more accurate than the conventional model. However, the new model takes much longer simulation time than the conventional model, which makes the estimation much difficult to accomplish.

One of the most effective design methods in the deep submicron technology would be the "floorplan oriented design method." A new HW/SW codesign approach based on this method is proposed to reduce the design time and effort, and improve the quality of design[73]. This approach consists of the following breakthrough points:

(1) Higher abstraction level of design.

(2) Higher reusability of pre-designed models.

(3) Accurate estimation of design quality.

(4) Optimization design assistance.

To obtain a more accurate estimation of design quality in an earlier stage of a design, floorplan oriented design method using Flexible Hardware Models (FHMs) will be effective. If an accurate design quality of a module can be utilized before instantiation, chip design can be proceeded without generating real design instance. We call this design method "Flexible Chip Design" [99], which will reduce design cost dramatically and chips will be designed concurrently for each design entity. Floorplan oriented design method will be easily realized if FHM provides a "flexible" layout at the physical level of design as well.

The concept of the FHM is the key to the proposed design methodology. The features of FHM can be summarized as follows:

(1) Parametalization.

(2) Instance Generation.

(3) Design Quality Estimation.

If the obtained estimation has a higher fidelity, better partitioning will result. Thus Flexible Chip Design can be realized. Even if a functionality is implemented by software, we can estimate the design quality of the software implementation. This uniform view of design quality will make our codesign methodology very powerful.

In order to overcome difficulties by using deep submicron technology, a new codesign approach based on PEAS concept must be addressed.

Bibliography

[1] Semiconductor Industry Association: "The National Technology Roadmap for Semiconductors Technology Needs 1997 Edition," 1997.

[2] J. A. Silberman: "Design of a 1.0 GHz 64-bit Integer Processor" *Proc. SASIMI'98*, 1998.

[3] B. A. Gieseke, R. L. Allmon, D. W. Bailey, B. J. Benschneider, S. M. Britton, J. D. Clouser, H. R. Fair III, J. A. Farrell, M. K. Gowan, C. L. Houghton, J. B. Keller, T. H. Lee, D. L. Leibholz, S. C. Lowell, M. D. Matson, R. J. Matthew, V. Peng : "A 600 MHz superscalar RISC microprocessor with out-of-order execution" *ISSCC Digest of Tech. Papers*, pp. 176-177, 1997.

[4] J. Schutz and R. Wallace: "A 450 MHz IA32 P6 family microprocessor" *ISSCC Digest of Tech. Papers*, pp. 236-237, 1998.

[5] N. Rohrer, C. Akrout, M. Canada, D. Cawthron, B. Davari, R. Floyd, S. Geissler, R. Goldblatt, R. Houle, P. Kartschoke, D. Kramer, P. McCormick, G. Salem, R. Schulz, L. Su, L. Whitney and J. H. Wuorinen : "A 480 MHz RISC microprocessor in a 0.12 *mum* L_{eff} CMOS technology with copper technology" *ISSCC Digest of Tech. Papers*, pp. 240-241, 1998.

[6] C. F. Webb, C. J. Anderson, L. Sigal, K. L. Shepard, J. S. Liptay, J. D. Warnock, B. Curran, B. W. Krumm, M. D. Mayo, P. J. Camporese, E. M. Schwarz, M. S. Farrell, P. J. Restle, R. M. Averill III, T. J. Siegel, W. V. Huott, Y. H. Chan, B. Wile : "A 400 MHz S/390 microprocessor" *IEEE Journal of Solid State Circuits*, Vol. 32, No. 11, pp. 1665-1675, 1997.

[7] H. P. Hofstee, S. H. Dhong, D. Meltzer, K. J. Nowka, J. A. Silberman, J. I. Burns, S. D. Posluszny and O. Takahashi : "Designing for a gigahertz" *IEEE Micro*, pp. 66-74, May-June 1998.

[8] D. Boerstler and K. Jenkins: "A phase-locked loop clock generator for a 1 GHz microprocessor" *Proc. 1998 Symp. on VLSI Circuits*, pp. 212-213, 1998.

[9] R. Heald, K. Shin, V. Reddy, I.-F. Kao, M. Khan, W. Lynch, G. Lauterbach, J. Petolino and J. H. Wuorinen : "64kB sum-addressed-memory cache with 1.6ns cycle and 2.6ns latency" *ISSCC Digest of Tech. Papers*, pp. 350-352, 1998.

[10] D. Heidel, Sang Dhong, P. Hofstee, M. Immediato, K. Nowka, J. Silberman and K. Stawiasz : "High speed serializing/de-serializing design-for-test method for evaluating a 1 GHz microprocessor" *Proc. 16th IEEE VLSI Test Symp.*, 1998.

[11] T. Shimizu, J. Korematu, M. Satou, H. Kondo, S. Iwata, K. Sawai, N. Okumura, K. Ishimi, Y. Nakamoto, M. Kumanoya, K. Dosaka, A. Yamazaki, Y. Ajioka, H. Tsubota, Y. Nunomura, T. Urabe, J. Hinata, K. Saitoh, J. H. Wuorinen: "A Multimedia 32b RISC Microprocessor with 16Mb DRAM," *IEEE International Solid-State Circuits Conference*, Digest of Tech. Paper, pp.216-217, 1996.

[12] J. Sato, M. Imai, T. Hakata, A. Y. Alomary and N. Hikichi: "An Integrated Design Environment for Application Specific Integrated Processor" *Proc. ICCD'91*, pp. 414-417, 1991.

[13] D. E. Thomas, J. K. Adams and H. Schmit: "A Model and Methodology for Hardware-Software Codesign" *IEEE Design & Test*, Vol. 10, No. 3, pp. 6-15, 1993.

[14] A. Kalavade and E. A. Lee: "A Hardware-Software Codesign Methodology for DSP Applications" *IEEE Design & Test*, Vol. 10, No. 3, pp. 16-28, 1993.

[15] R. K. Gupta and G. De Micheli: "Hardware-Software Cosynthesis for Digital Systems" *IEEE Design & Test*, Vol. 10, No. 3, pp. 29-41, 1993.

[16] R. Ernst, J. Henkel and T. Benner: "Hardware-Software Cosynthesis for Microcontroller" *IEEE Design & Test*, Vol. 10, No. 4, pp. 64-75, 1993.

[17] H. Yasuura, S. Nakamura, H. Tomiyama and H. Akaboshi: "Hardware-Software Codesign with a Soft-Core Processor" *Proc. SASIMI'95*, pp. 79-84, 1995

[18] I. Huang, B. Holmer and A. M. Despain: "ASIA: Automatic Synthesis of Instruction-Set Architectures" *Proc. SASIMI'93*, pp. 13-22, 1993.

[19] H. Akaboshi and H. Yasuura: "COACH: A Computer Aided Design Tool for Computer Architects" *Trans. IEICE*, Vol. E76-A, No. 10, pp. 1760-1769, 1993.

[20] J. Sato, A. Y. Alomary, Y. Honma, T. Nakata, A. Shiomi, N. Hikichi and M. Imai: "PEAS-I: A Hardware/Software Codesign System for ASIP Development" *Trans. IEICE*, Vol. E77-A, No. 3, pp. 483-491, 1994.

[21] I. Pyo, C. Su, I. Huang, K. Pau, Y. Koh, C. Tsui, H. Chen, G. Cheng, S. Liu, S. Wu and A. M. Despain: "Application-driven Design Automation for Microprocessor Design" *Proc. 29th DAC*, pp. 512-517, 1992.

[22] A. Y. Alomary, M. Imai, J. Sato and N. Hikichi: "An Integer Programming Approach to Instruction Set Selection Problem" *Trans. IEICE*, Vol. E76-A, No. 10, pp. 1849-1857, 1993.

[23] N. N. Binh, M. Imai, A. Shiomi, N. Hikichi and J. Sato: "An Efficient Scheduling Algorithm for Pipelined Instruction Set Processor and Its Application to ASIP Hardware/Software Codesign" *Trans. IEICE*, Vol. E78-A, No. 3, pp. 353-362, 1995.

[24] N. N. Binh, M. Imai, A. Shiomi and N. Hikichi: "A Hardware/Software Partitioning Algorithm for Designing Pipelined ASIPs with Least Gate Count," *Proc. of DAC'96*, pp. 527-532, 1996.

[25] N. N. Binh, M. Imai and Y. Takeuchi: "A Performance Maximization Algorithm to Design ASIPs under the Constraint of Chip Area Including RAM and ROM," *Proc. of ASP-DAC'98*, pp. 367-372, 1998.

[26] I. Huang and A. M. Despain: "High Level Synthesis of Pipelined Instruction Set Processors and Back-End Compilers" *Proc. 29th DAC*, pp. 135-140, 1992.

[27] H. Tomiyama, H. Akaboshi and H. Yasuura: "Compiler Generator for Hardware/Software Codesign" *Proc. 2nd APCHDL*, pp. 267-270, 1994.

[28] H. Akaboshi and H. Yasuura: "Automatic Generation of Instruction Level Simulation Model for Processor from RTL Level Description" (in Japanese) *Trans. IEICE*, Vol. J78-A, No. 8, pp. 919-928, 1995.

[29] J. L. Hennessy and D. A. Patterson: Computer Architecture A Quantitative Approach. *2nd Edition*, Morgan Kaufmann Publishers, 1996.

[30] F. de Schepper, K. Yamazaki, M. Imai and J. Sato: "Application of ASIC-Technology to Mechatronics Control: Development of the Flexible Servo Peripheral Chip" *Annals of CIRP*, Vol. 37/1/1988, pp. 389-392, 1988.

[31] T. Kimura, J. Sato, M. Imai, F. de Schepper and K. Yamazaki: "The Design of a FSP-3 by using Silicon Compiler GENESIL" (in Japanese) *IEICE Tech. Report*, Vol. VLD89-107, pp. 75-81, 1990.

[32] J. Sato, T. kimura, M. Imai, F. de Schepper, K. Yamazaki, M. Nagase and S. Yamamoto: "The Architecture of a Flexible Servo Motor Control Processor - FSP-3 -" *Trans. IEICE*, Vol. E73, No. 4, pp. 513-515, 1990.

[33] R. Stallman: "Using and Porting GNU C Compiler, Version 1.40" *Free Software Foundation Inc.*, 1991.

[34] Y. Nakamura, K. Oguri, R. Nomura, A. Nagoya and M. Yukishita: "RTL Behavioral Description Language SFL" *Trans. IEICE*, Vol. J72-A, No. 10, pp. 1579-1593, 1990.

[35] NTT Data Communications: "PARTHENON User's Manual" 1989.

[36] K. Oguri, Y. Nakamura, R. Nomura, A. Nagoya and M. Yukishita: "PARTHENON: Perfect Harmony between Behavioral Language SFL and Synthesizer" *Proc. 4th KARUIZAWA Workshop on Circuits and Systems*, pp. 198-203, 1991.

[37] http://www-mount.ee.umn.edu/mcerg/software.html

[38] M. D. Hill and A. J. Smith: "Experimental Evaluation of On-Chip Microprocessor Cache Memories," *Proc. of 11th International Symposium on Computer Architecture*, pp. 158-166, 1984.

[39] ftp://havefun.stanford.edu/pub/jpeg/JPEGv1.2.1.tar.Z

[40] ftp://havefun.stanford.edu/pub/mpeg/MPEGv1.2.1.tar.Z

[41] ftp://havefun.stanford.edu/pub/p64/P64v1.2.2.tar.Z

[42] R. Composano and W. Rosenstiel: "Synthesizing Circuits from Behavioral Description" *IEEE Trans. CAD*, Vol. 8, No. 2, pp. 171-180, 1989.

[43] R. Composano and W. Wolf: "High-Level VLSI Synthesis" *Kluwer Academic Publishers*, 1991.

[44] G. De Micheli and D. C. Ku: "HERCULES-A CAE/CAD Tool Development" *Proc. 25th DAC*, pp. 483-488, 1988.

[45] S. Devadas and A. R. Newton: "Algorithms for Hardware Allocation in Data Path Synthesis" *IEEE Trans. CAD*, Vol. 8, No. 7, pp. 768-781, 1987.

[46] C. H. Gebotys and M. I. Elmasry: "VLSI Design Synthesis with Testability" *Proc. 25th DAC*, pp. 16-21, 1988.

[47] C. H. Gebotys and M. I. Elmasry: "Optimal VLSI Architectural Synthesis: Area, Performance and Testability" *Kluwer Academic Publishers*, 1992.

[48] M. Imai: "Trend of Hardware Description Language and High-Level Synthesis" *Proc. The 4th KARUIZAWA Workshop on Circuits and Systems*, pp. 174-179, 1991.

[49] T. H. Meng: "Synchronization Design for Digital Systems" *Kluwer Academic Publishers*, 1991.

[50] Z. Peng: "CAMAD: A Unified Data Path/Control Synthesis" *Proc. IFIP Working Conf. on Design Methodologies for VLSI and Computer Architecture*, pp. 53-67, 1988.

[51] H. Trickey: "Flamel: A High-Level Hardware Compiler" *IEEE Trans. CAD*, Vol. 6, No. 2, pp. 259-269, 1987.

[52] D. R. Coelho: "The VHDL Handbook" *Kluwer Academic Publishers*, 1989.

[53] IEEE: "IEEE Standard VHDL Language Reference Manual" *IEEE*, 1988.

[54] R. Joobbani: "Requirements of a VHDL-Based Design Environment" *Proc. 4th KARUIZAWA Workshop on Circuits and Systems*, pp. 180-186, 1991.

[55] S. S. Leung and M. Shanblatt: "ASIC System Design with VHDL: A Paradigm" *Kluwer Academic Publishers*, 1989.

[56] Open Verilog International: "Verilog Hardware Description Language Reference Manual" *Draft Release 0.1*, Open Verilog International,1991.

[57] D. Rich: "Mixed-Level Simulation and Design Synthesis Based on the Verilog Hardware Description Language" *Proc. 4th KARUIZAWA Workshop on Circuits and Systems*, pp. 193-197, 1991.

[58] D. Thomas: "The Verilog Hardware Description Language" *Kluwer Academic Publishers*, 1990.

[59] T. Hoshino: "HSL-FX: A Unified Language for VLSI Design" *Proc. of CHDL85*, 1985.

[60] O. Karatsu: "UDL/I: A Hardware Description Language Standard for Logic Synthesis Age" *Trans. IEICE*, Vol. J74-A, No. 2, pp. 170-178, 1991.

[61] UDL/I Committee: "UDL/I language Reference Manual Version 2.0.2" *Japan Electronic Industry Development Association*, 1993.

[62] D. Gajski, A. Wu, N. Dutt and S. Lin: "High-Level Synthesis - Introduction to Chip and System Design -" *Kluwer Academic Publishers*, 1992.

[63] M. Imai: "Estimation and Synthesis of Hardware" (in Japanese) *Journal of IPSJ*, Vol. 36, No. 7, pp. 614-619, 1995.

[64] H. Yasuura: "Basic Software Codesign" (in Japanese) *Journal of IPSJ*, Vol. 36, No. 7, pp. 620-626, 1995.

[65] T. Miyazaki: "Hardware/Software Codesign for Digital Signal Processing" (in Japanese) *Journal of IPSJ*, Vol. 36, No. 7, pp. 627-632, 1995.

[66] G. De Micheli: "Computer-Aided Hardware-Software Codesign" *IEEE Micro*, Vol. 14, No. 4, pp. 10-16, 1994.

[67] P. Chou, R. Ortega and G. Borriello: "Synthesis of the Hardware/Software Interface in Microcontoroller-Based Systems" *Proc. ICCAD'92*, pp. 488-495, 1992.

[68] D. D. Gajski and F. Vahid: "Specification and Design of Embedded Hardware-Software Systems" *IEEE Design & Test*, Vol. 12, No. 1, pp. 53-67, 1995.

[69] J. Sato and M. Imai: "A Study on the Application Specific Microprocessor Design Environment" *Proc. SASIMI'90*, pp. 88-94, 1990.

[70] J. Sato, M. Imai, T. Hakata and N. Hikichi: "Proposal of a New Design Environment for Application Specific Integrated Processor: IDEAS" *Trans. IEICE*, Vol. E74, No. 5, pp. 1014-1016, 1991.

[71] J. Sato, N. Hikichi, A. Shomi and M. Imai: "Effectiveness of a HW/SW Codesign System PEAS-I in the CPU Core Design" *Proc. APCHDL'94*, pp. 259-262, 1994.

[72] J. Sato, N. Hikichi, T. Nakata, Y. Honma, A. Shiomi and M. Imai: "Effectiveness Evaluation of a Hardware/Software Codesign System PEAS-I" (in Japanese) *Proc. 7th KARUIZAWA Workshop on Circuits and Systems*, pp. 309-314, 1994.

[73] M. Imai, A. Shiomi, Y. Takeuchi, J. Sato and Y. Honma: "Hardware/Software Codesign in the Deep Submicron Era" *Proc. IWLAS'96*, pp. 236-248, 1996.

[74] T. Morifuji, Y. Takeuchi, J. Sato and M. Imai: "Flexible Hardware Model: Implementation and Effectiveness" *Proc. SASIMI'97*, pp. 83-89, 1997.

[75] I. Huang and A. M. Despain: "Synthesis of Instruction Set for Pipelined Microprocessors" *Proc. 31st DAC*, pp. 5-11, 1994.

[76] I. Huang and A. M. Despain: "Hardware/Software Resolution of Pipeline Hazards in Pipeline Synthesis of Instruction Set Processor" *Proc. ICCAD'93*, pp. 594-599, 1993.

[77] M. D. Hill and A. J. Smith: "Evaluating Associativity in CPU Caches," *IEEE Trans. on Computers*, Vol. 38, No. 12, pp.1612–1630, 1989.

[78] A. Agarwal, P. Chow, M. Horowitz, J. Acken, A. Salz and J. Hennessy: "On-Chip Instruction Caches for High Performance Processors," *Advanced Research in VLSI*, pp. 1–24, 1987.

[79] A. J. Smith: "Line (Block) Size Choice for CPU Cache Memories," *IEEE Trans. on Computers*, Vol. C-36, No. 9, pp.1063–1075, 1987.

[80] A. Agarwal, M. Horowitz and J. Hennessy: "An Analytical Cache Model," *ACM Trans. on Computer Systems*, Vol. 7, No. 2, 1989.

[81] R. T. Short and H. M. Levy: "A Simulation Study of Two-Level Caches," *Proc. of the 15th Annual International Symposium on Computer Architecture, pp.81–89, 1988.*

[82] K. Akeley: "The Silicon Graphics 4D/240GTX Superworkstation," *IEEE Computer Graphics and Applications*, Vol. 9, No. 4, pp.41–70, 1986.

[83] S. T. Fu and M. J. Flynn: "Optimal on-chip cache hierarchy synthesis with scaling of technology," *In Proceedings of 1996 IEEE International Phoenix Conference on Computers and Communications*, pp.129-135, 1996.

[84] N. P. Jouppi and S. J. E. Wilton: "Tradeoffs in Two-Level On-Chip Caching," *Proc. of 21st Annual International Symposium on Computer Architecture*, pp. 34-45, 1995.

[85] R. Hundal and V. G. Oklobdzija: "Determination of Optimal Sizes for a First and Second Level SRAM-DRAM On-Chip Cache Combination," *Proceedings*

IEEE INTERNATIONAL CONFERENCE ON Computer Design: VLSI in Computer & Processors, pp. 60-64, 1994.

[86] S. T. Fu and M. J. Flynn: "Optimal on-chip cache hierarchy synthesis with scaling of technology," *Proc. of 1996 IEEE International Phoenix Conference on Computers and Communications*, pp. 129-135, 1996.

[87] J. Sato, Y. Takeuchi, M. Imai, K. Yoshioka and A. Shiomi: "Evaluation of a Hierarchical On-Chip Memory System for ASIPs" *TECHNICAL REPORT OF IEICE*, pp. 17-24, 1996.

[88] K. Yoshioka, J. Sato, Y. Takeuchi and M. Imai: "A Performance Optimization Method for an On-Chip Two Level Cache Memory System" *Proc. SASIMI'97*, pp. 98-104, 1997.

[89] J. Sato, K. Yoshioka, Y. Takeuchi and M. Imai: "A Configuration Optimization Method for On-Chip Two-Level Cache Memory based on Memory Access Sequence Analysis" *Submitted to IPSJ*, (Sept. 1998).

[90] D. Kirovski, C. Lee, M. Potkonjak and W. Mangione-Smith: "Application-Driven Synthesis of Core Based Systems" *Proc. ICCAD'97*, pp. 104-107, 1997.

[91] M. J. Flynn: "Computer Architecture - Pipelined and Parallel Processor Design" *Jones & Bartlett Publishers*, 1995.

[92] P. R. Panda, N. Dutt and A. Nicolau: "Exploiting Off-Chip Memory Access Modes in High-Level Synthesis" *Proc. ICCAD'97*, pp. 333-340, 1997.

[93] P. R. Panda, N. Dutt and A. Nicolau: "Efficient Utilization of Scratch-Pad Memory in Embedded Processor Applications" *ED&TC'97*, 1997.

[94] P. R. Panda: "Data Cache Sizing for Embedded Processor Applications" *UCI-ICS TR97-30*, 1997.

[95] P. R. Panda, N. Dutt and A. Nicolau: "Memory Data Organization for Improved Cache Performance in Embedded Processor Applications" *ACM Trans. DAES*, Vol. 2, No. 4, 1997.

[96] N. Dutt: "Memory Organization and Exploration for Embedded Systems-on-Silicon" *Proc. ICVC'97*, 1997.

[97] H. Tomiyama and H. Yasuura: "Optimal Code Placement of Embedded Software for Instruction Cache" *Proc. ED&TC'96*, 1996.

[98] S. Bakshi and D. D. Gajski: "A Memory Selection Algorithms for High-Performance Pipelines" *Proc. EuroDAC'95*, 1995.

[99] M. Muraoka: "EDA Technology Direction Towards 2010" *Proc. SASIMI'98*, 1998.

[100] H. Yasuura, H. Tomiyama, A. Inoue, and F. N. Eko: "Embedded System Design Using Soft-Core Processor and Valen-C" *Proc. APCHDL'97*, pp. 121-130, 1997.

List of Major Publications of the Author

Journal Papers (Refereed)

(1) J. Sato, T. Kimura, M. Imai, F. de Schepper, K. Yamazaki, M. Nagase, S. Yamamoto: "The Architecture of a Flexible Servo Motor Control Processor: FSP-3," Trans. of IEICE, Vol. E73, No. 4, pp. 513-515, Apr. 1990.

(2) J. Sato, M. Imai, T. Hakata and N. Hikichi: "Proposal of a New Design Environment for Application Specific Integrated Processor," Trans. of IEICE, Vol. E74, No. 5, pp. 1014-1016, May 19[illegible]

(3) J. Sato, A. Y. Alomary, Y. Honma, T. Nakata, A. Shiomi, N. Hikichi and M. Imai: "PEAS-I: A Hardware/Software Co-design System for ASIP Development," Trans. of IEICE, Vol. E77-A, No. 3, pp. 483-491, Mar. 1994.

(4) J. Sato, K. Yoshioka, Y. Takeuchi and M. Imai: "A Performance Optimization Method for On-chip Two-Level Cache Memory based on Memory Access Sequence Analysis," Submitted to IPSJ (Sept. 1998).

International Conference Papers (Refereed)

(1) J. Sato and M. Imai: "A Study on the Application Specific Microprocessor Design Environment," Proc. of the Synthesis and Simulation Meeting and International Interchange (SASIMI'90), pp. 88-94, Oct. 1990.

(2) J. Sato, M. Imai, T. Hakata and N. Hikichi: "An Integrated Design Environment for Application Specific Integrated Processor," Proc. of International

Conference on Computer Design (ICCD'91), pp. 414-417, Oct. 1991.

(3) J. Sato, N. Hikichi, A. Shiomi and M. Imai: "Effectiveness of a HW/SW Codesign System PEAS-I in the CPU Core Design," Proc. of 2nd Asia Pacific Conference on Hardware Description Languages (APCHDL'94), pp. 259-262, Oct. 1994.

(4) K. Yoshioka, J. Sato, Y. Takeuchi and M. Imai: "A Performance Optimization Method for an On-Chip Two Level Cache Memory System," Proc. of Synthesis and Simulation Meeting and International Interchange (SASIMI'97), pp. 98-104, Dec. 1997.

National Conference Papers (Refereed)

(1) J. Sato, K. Fukuda, M. Ichida and M. Imai: "A Study on the Application Specific Microprocessor Design Environment," Proc. of 3rd Karuizawa Workshop on Circuits and Systems, pp. 74-81, Apr. 1990.

(2) J. Sato, A. Y. Alomary, Y. Honma, T. Nakata, T. Hakata, N. Hikichi and M. Imai: "Current Status of a Hardware/Software Codesign System PEAS-I for ASIP Development," Proc. of 6th Karuizawa Workshop on Circuits and Systems, pp. 513-518, Apr. 1993.

(3) J. Sato, N. Hikichi, T. Nakata, Y. Honma, A. Shiomi and M. Imai: "Effectiveness Evaluation of a Hardware/Software Codesign System PEAS-I," Proc. of 7th Karuizawa Workshop on Circuits and Systems, pp. 309-314, Apr. 1994.