Researches on fabrication of low-energy heterogeneous multi-core processors

by

Tomoyuki Nakabayashi
(411D051)

The Division of Systems Engineering,
Graduate School of Engineering, Mie University.

Tsu, Mie

March, 2014

APPROVED BY:

Dr. Toshio Kondo (Chair of Advisory Committee)
Dr. Yoshikatsu Ohta
Dr. Kazuo Mori

ABSTRACT

Since energy consumption and heat density are growing problems in high-performance processors as well as embedded processors, lots of latest researches on computer systems aim at enhancing energy-efficiency of processors. One of leading energy-efficient approaches, single-instruction-set-architecture (single-ISA) heterogeneous multi-core processor comprised of microarchitecturally diverse cores (differently-designed in microarchitecture level), gets much attention from the researchers. Two key challenges in the heterogeneous paradigm are (1) the development of energy-efficient processor core highlighting finer heterogeneity in an application phase, and (2) the design automation of the entire single-ISA heterogeneous multi-core processor. The author studies basic circuit and architecture for (1), and develops a processor design environment for (2) as described below in detail:

(1) This work proposes a combinational approach across two different fields to develop a low-energy processor core, i.e., circuit-level (low-energy D-flip-flops) and microarchitecture-level (variable stages pipeline) approaches.

Circuit-level approach: D-flip-flops play an important role in a processor chip because the delay, area, and power consumption of D-flip-flops drastically affect the performance of the processor. This work proposes two types of novel D-flip-flop which adopt semi-static and true-single-phase clock (TSPC) schemes. One is called double split-output semi-static TSPC D-flip-flop (DSSTSPC D-flip-flop) emphasizing short circuit delay by a novel front-end composed of parallelized split-output latches. The other, single split-output semi-static TSPC D-flip-flop (SSSTSPC D-flip-flop), takes a special focus on low-energy operation by removing a part of DSSTSPC D-flip-flop. The former shortens the circuit delay by 5% compared with a conventional low-energy D-flip-flop without increase in the energy and layout area. The latter achieves 31% smaller layout area and 30% lower energy consumption with up to 8% performance degradation compared with the conventional D-flip-flop.

Microarchitecture-level approach: Modern processors widely employ dynamic voltage and frequency scaling (DVFS) technique which dynamically scales the supply voltage and clock frequency in accordance with workload on the processor. Although DVFS is effective for energy saving, it suffers from its large overhead when we intend a temporally fine-grain energy optimization. To compensate for DVFS, a variable stages pipeline (VSP) architecture is

proposed. VSP reduces the energy consumption by dynamically varying the pipeline depth, instead of the supply voltage, depending on instruction-level behavior in a running program. Since the penalty for a pipeline scaling is small enough to reduce the energy consumption at tens or hundreds clock cycles, VSP can save the energy consumption at finer-grain period than DVFS. This thesis proposes a fine-grain depth-changing method which can be implemented by a trivial FIFO buffer to detect processor workload, and presents its chip fabrication on a 180 nm technology. Evaluation results using the fabricated VSP chip show that the VSP reduces the energy consumption by 34% to 48% at fine-grained low-energy operation insertion which is impossible with DVFS. Moreover, we adopt a special cell called latch D-flip-flop selector-cell (LDS-cell) into VSP processor to further reduce the energy consumption under folded pipeline structure. This thesis reveals that inserting LDS-cells makes VSP processor consume 13% less energy on a fabricated chip. (2) This thesis also presents a development environment that improves research productivity by automatic design generation and co-simulation framework, especially fabrication and prototyping through a standard ASIC design flows.

Automatic design generation: Because a single-ISA heterogeneous multi-core consists of microarchitecturally diverse cores to streamline the execution of diverse program phases, the design and verification effort is multiplied by the number of employed core types. The increased design effort impedes development of heterogeneous multi-core processors. N. K. Choudhary et al. develop a toolset, called FabScalar, for automatically composing the synthesizable designs of arbitrary cores. Although using FabScalar helps mitigate the design effort, the design effort for diverse cache systems and a shared bus still exists as a barrier in the development of heterogeneous multi-core processor. This work proposes FabHetero which is composed of three design automation tools: FabScalar, FabCache, and FabBus for automatically composing diverse cores, cache systems, and flexible shared bus, respectively. FabHetero project sets a goal of fabricating heterogeneous multi-core processor chips in a short time, and this work is the first attempt to automate the entire heterogeneous multi-core design. The author confines the microarchitectural diversity into a superset code that enables users to use a single universal design of heterogeneous multi-core processor; however, the footprint of each design is the desired configuration. FabCache automatically designs many caches that satisfy the requirements from modern superscalars and differ in cache dimensions. FabBus automates generating a flexible shared bus which connects the arbitrary number of caches with desired cache coherence

protocol.

Co-simulation framework: Furthermore, FabHetero framework includes a practical processor co-simulation framework for not only RTL simulation but also gate/transistor level simulation, and even fabricated chip evaluation/validation. Our framework addresses the following two challenges: system call emulation and sampled execution. Both mechanisms are commonly used only in software processor simulators; therefore, this work introduces these mechanisms into standard ASIC design flows using off-chip system call emulator and checkpoint mechanism. Processor design can remain unchanged from its pure specification (no extra I/Os and hardware is needed) because the proposed mechanisms exploit general instructions inherent in processor.

This work provides a great step: automatic generation of an entire processor design involving a superscalar core, cache system, and bus system and its fabrication in shortened design time using the co-simulation framework. This helps researchers fabricate their novel processor chips by much less effort.

Contents

List of Figures

List of Tables

1 Introduction

Since energy consumption and heat density are growing problems in high-performance processors as well as embedded processors, lots of latest researches on computer systems aim at enhancing energy efficiency of processors. The purpose of this work is to develop energy-efficient processors. Many techniques to improve the energy efficiency have been developed in wide research fields of device, circuit, microarchitecture, and software. This thesis mainly focuses on development of microarchitectural low-energy technique because microarchitectural approaches have an advantage in that they do not depend on specific device technology and can be widely used.

As one of leading low-energy approaches, single-instruction-set-architecture (single-ISA) heterogeneous multi-core processor, which is comprised of microarchitecturally diverse cores (differently-designed in microarchitecture level), gets much attention from the researchers [1–3]. Heterogeneous multi-core processors streamline the execution of diverse programs and program phases using differently-designed core types. Two key challenges for emerging low-energy processors in the heterogeneous paradigm are (1) the development of energy-efficient processor core highlighting finer heterogeneity in an application phase, and (2) the design automation of the entire single-ISA heterogeneous multi-core processor. The author studies basic circuit scheme and architectural approach for (1), and develops a processor design environment for (2).

1.1 Development of Energy-efficient Processor Core

While a heterogeneous multi-core processor contains diverse core types to enhance energy efficiency, each core design is still required to be energy efficient. This work proposes combination across two different level approaches to de-

velop a low-energy processor core, i.e., circuit-level (low-energy D-flip-flops) and microarchitecture-level (variable stages pipeline) approaches.

1.1.1 Proposing Low-energy D-flip-flop

In the design of digital circuit, D-flip-flops have a larger impact on the circuit speed, area and power consumption than other standard cells. The timing of a design is significantly dependent on the speed of the D-flip-flops, particularly in deep-pipelined designs. The design trend increases the pipeline stages for higher-performance processors, and this has also prompted an increase in the number of D-flip-flops in a chip [4]. D-flip-flops dominate 38 to 48% of the graphic processing LSI area [5], and 22% of the area in variable stages pipeline processor (the author's another research project). In many digital circuits, the power consumption in the clock system, which includes the clock network and D-flip-flops, may be 20 to 40% of the total chip power consumption [6, 7]. These aspects represent improving the performance of D-flip-flops is of value to enhance processor performance.

In order to achieve both high performance and low energy, a high-performance semi-static true-single-phase clocking D-flip-flop (HSTSPC D-flip-flop) is proposed [5]. The HSTSPC D-flip-flop is based on true-single-phase clock (TSPC) scheme, so that it can achieve higher speed and lower energy than conventional D-flip-flops. In addition, the HSTSPC D-FF adopts a semi-static structure, that is suitable for conventional digital circuit design. However, because there is a difference between the number of NMOS and PMOS transistors, an area-efficient layout is difficult. In addition, the output of front-end does not fully drive to high level because it is pulled up by NMOS transistor.

This work proposes two novel D-flip-flops which improve HSTSPC D-flip-flop. One is double split-output semi-static TSPC D-flip-flop (DSSTSPC

D-flip-flop) emphasizing short circuit delay by a novel front-end comprised of parallelized split-output latches. Proposed scheme solves the layout problem of HSTSPC D-flip-flop using the same number of NMOS and PMOS transistors. Furthermore, the novel front-end fully and quickly drives its output, consequently, the D-flip-flop improves operation speed. The other, single split-output semi-static TSPC D-flip-flop (SSSTSPC D-flip-flop), takes a special focus on low-energy operation by removing a part of DSSTSPC D-flip-flop. Both D-flip-flops achieve higher energy-efficiency than conventional D-flip-flops and are suitable for standard cell based design widely used in design of embedded processors. Proposed D-flip-flops help in developing energy-efficient processor cores.

1.1.2 Enhancement of Variable Stages Pipeline Architecture

On a heterogeneous multi-core core processor, matching instruction-level behavior in programs to differently-designed cores improves energy-efficiency; a scheduling method allocates a program or program phase (even if a program is running) to a suitable processor. This may cause frequent migrations between cores; however, a migration incurs a significant overhead, i.e., passing architectural state (program counter, register file, etc.) and cache state. The migration penalty motivates us to develop processor cores highlighting finer heterogeneity in a program phase. Modern processors widely employ dynamic voltage and frequency scaling (DVFS) technique which dynamically scales the supply voltage and clock frequency in accordance with workload on the processor [8, 9]. Lowering the supply voltage is of effectiveness for energy reduction because the energy consumption depends on the square of the supply voltage. However, DVFS suffers from its large overhead when the author intends a temporally fine-grain energy optimization [10].

To optimize energy consumption at finer-grain interval than DVFS's interval, a variable stages pipeline (VSP) architecture is proposed. VSP reduces the energy consumption by dynamically varying the pipeline depth, instead of the supply voltage, depending on behavior of a running program and program phase. Since the penalty for a pipeline scaling is small enough to reduce the energy consumption at tens or hundreds clock cycles, VSP can save the energy consumption at two or three orders of magnitude finer-grain than DVFS. The author proposes a fine-grain depth-changing method with can be implemented by a trivial FIFO buffer to detect processor workload, and presents its chip fabrication on a 180 nm technology.

VSP technique can be used along with proposed D-flip-flops to further improve energy efficiency.

1.2 Design Effort Reduction for Fabricating Heterogeneous Multi-core Processors

In the heterogeneous era, processor designers will suffer huge design effort because they must develop differently-designed cores, suitable cache system for each core, and a shared bus on a die. This situation suggests to us that the design automation of diverse cores, caches, and shared bus is strongly required. The author frames a novel framework to automatically generate the entire heterogeneous multi-core processors in a design space. This thesis also presents a practical co-simulation mechanism that improves research productivity, especially fabrication and prototyping through a standard ASIC design flows.

1.2.1 Automatic Design Generation of the Entire Heterogeneous Multi-core Processor

Because a single-ISA heterogeneous multi-core consists of microarchitecturally diverse cores to streamline the execution of diverse programs and program phases, the design and verification effort is multiplied by the number of employed core type. The increased design effort impedes development of heterogeneous multi-core processors. N. K. Choudhary et al. develop a toolset, called FabScalar, for automatically composing the synthesizable register-transfer-level (RTL) designs of arbitrary cores within a superscalar template [11]. Although using FabScalar helps mitigate the design effort, design effort for diverse cache systems and a flexible shared bus still exists as a barrier in the development of a whole heterogeneous multi-core processor. Cache system is widely recognized as a key factor impacting on the performance, area, and energy. Therefore, each cache system should be dedicated for each processor core for targeted program and program phase. In addition, the shared bus in a heterogeneous multi-core processor requires a high flexibility because it must connect diverse cache designs. These two aspects incur a high design effort even after automating core development. To automatically design the entire heterogeneous multi-core processors, the author proposes FabHetero which is composed of three design automation tools: FabScalar, FabCache, and FabBus for developing diverse cores, cache systems, and flexible shared bus, respectively. FabHetero project sets a goal of fabricating heterogeneous multi-core processor chips in a short time, and this work is the first attempt to automate the entire heterogeneous multi-core design in a design space.

1.2.2 Co-simulation Framework for Streamlining Processor Development

As multi-core architecture has become commonly used to improve processor performance, designing a state-of-the-art multi-core chip in a short time has become essential for processor research. A development environment that contains useful mechanisms and can be used throughout the entire processor research provides efficient infrastructure to researchers. FabHetero framework includes a practical processor co-simulation framework for not only RTL simulation but also gate/transistor level simulation, and even fabricated chip evaluation/validation with an LSI tester. The co-simulation framework addresses the following two challenges: system call emulation and sampled execution. Both mechanisms are commonly used only in software processor simulators; therefore, this work introduces these mechanisms into standard ASIC design flows. These mechanisms effectively reduce the design time in FabHetero project.

1.3 Thesis Contributions

This thesis is mainly composed of four works: each work addresses different challenge but all works head to fabrication of low-energy heterogeneous multi-core processors. Each contribution to the goal is explicitly described below.

- DSSTSPC D-flip-flop, targeting a high speed operation, shortens the circuit delay by 5% compared with a conventional low-energy D-flip-flop without increase in the energy and layout area [12]. The D-flip-flop also solves imbalanced scheme of the conventional D-flip-flop. This aspect enables designers to easily optimize the layout without a special layout model. This D-flip-flop can be used as a component in processor cores

intended to achieve high performance on a heterogeneous multi-core processor.

- SSSTSPC D-flip-flop, power and area-efficient design, achieves 31% smaller layout area and 30% lower energy consumption than the conventional D-flip-flop with up to 8% performance degradation [12]. Processor cores emphasizing low-energy consumption rather than high performance can employ the D-flip-flop instead of DSSTSPC D-flip-flop.

- This work presents the detailed energy evaluation and hardware cost to implement VSP architecture into a processor core. The author fabricated a VSP processor chip on a 180 nm technology. The chip fabrication clarifies that the hardware cost to adopt VSP approach is reasonable considering obtained effectiveness. Evaluation result using the fabricated VSP chip shows that the VSP reduces the energy consumption by 34% to 48% at fine-grained interval at which DVFS cannot be used [13, 14].

- This work proposes how to optimize the trade-off between energy and performance using VSP architecture for fine-grain heterogeneity in a program phase. Proposed depth-changing controller enables VSP to change its pipeline depth to a suitable depth for applications [15–17].

- A special cell called latch D-flip-flop selector-cell (LDS-cell) is adopted into VSP processor to further reduce the energy consumption under folded pipeline structure. LDS-cell prevents unnecessary signal transitions from propagating through combinational circuits. The author confirms that inserting LDS-cells makes VSP processor consume 13% less energy. The comparison between transistor-level simulation and fabricated chip evaluation reveals that an actual implementation of

VSP on a die has larger impact on energy saving, 5% energy saving on the simulation [18, 19].

- The author frames a framework, called FabHetero, for automatically generating the entire heterogeneous multi-core processors [20]. FabHetero consists of three design automation tools: FabScalar, FabCache, and FabBus for developing diverse cores, cache systems, and flexible shared bus, respectively. This work defines the design space of FabHetero and provides it as a research infrastructure.

- FabCache automatically generates diverse cache systems in heterogeneous multi-core processors. FabCache parameterizes many design diversity: cache hierarchy, cache dimensions, specific designs, and interface design [23].

- FabBus is a tool based on AMBA protocol to design flexible shared bus systems for heterogeneous multi-core processors. FabBus connects arbitrary number of caches with desired cache coherency protocol [24].

- The development of a practical processor co-simulation environment streamlines chip fabrication of processors on standard ASIC design flow [21,22]. This thesis addresses the following two challenges: system call emulation and sampled execution for not only RTL simulation but also gate/transistor level simulation, and even fabricated chip evaluation/validation with an LSI tester. Both mechanisms are commonly used only in software processor simulators; therefore the author introduces these mechanisms into standard ASIC design flows.

1.4 Thesis Organization

The rest of this thesis is organized as follows.

Chapter 2 describes background information related to this thesis. The chapter discusses process technology and low-energy technique trends. Chapter 3 provides the detail description of the low-energy D-flip-flops including evaluation results of energy, circuit delay, and layout area. It introduces D-flip-flop classification and related work. Chapter 4 elaborates on variable stages pipeline architecture. The chapter includes explanation of other variable-depth pipeline architectures, proposed depth-changing control method, detailed specification of fabricated VSP chip, and evaluation results using the chip. Chapter 5 describes FabHetero project including the specifications of FabCache and FabBus. It also introduces FabScalar developed by North Carolina State University, collaborator in FabHetero project. Chapter 6 presents co-simulation framework used in FabHetero project. The chapter also presents that proposed co-simulation framework does not depend on the microarchitecture of a processor; therefore, proposed approaches can be widely used in processor design projects. Chapter 7 summarizes this thesis and gives future work.

2 Background

This chapter provides background on this thesis: why single-ISA heterogeneous multi-core processor attracts much attention. The historical trend of processor architecture is classified into three eras in historical order of device technology and processor architecture in the following Sections. Section 2.1 describes the first era: improvement of single-core/thread performance. Section 2.2 presents multi-core technologies following from the limitation of improving single-core chip performance. Section 2.3 provides the trend of heterogeneous multi-core architecture. Section 2.4 summarizes the requirements for developing low-energy single-ISA heterogeneous multi-core processors targeted in this work.

2.1 Improvement of Single-core Performance

Moore's Law [25], the doubling of transistor density every 18 months, has been a fundamental driver of performance enhancement. Figure 2.1 shows the growing transistor integration for Intel processors. In the past three decades, transistors have become smaller, faster and more energy-efficient every technology node along with Dennard scaling [26]: transistors get smaller *but the power density is constant*. With Dennard scaling theory, processor architects can use a larger number of transistors with the same energy consumption on the same die area. For this reason, processor architects have improved the single-thread performance by adding more hardware for complex microarchitecture such as pipeline, superscalar, and out-of-order execution to exploit instruction level parallelism (ILP). Superscalar and out-of-order execution are the core ideas of modern computing for mobile computers, personal computers, and servers. A superscalar extracts ILP by analyzing the dependency between instructions and scheduling the execution order. A new superscalar

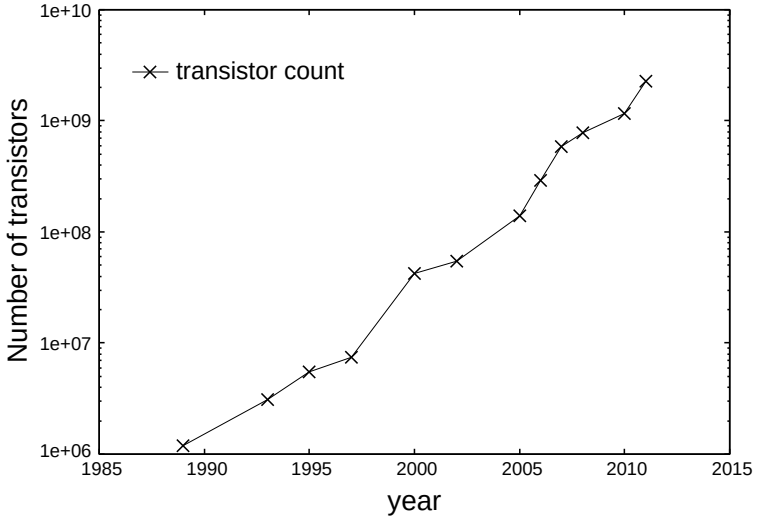

Figure 2.1: Impact of technology scaling on transistor count of Intel's processor.

processor can execute an existing binary program compiled for the same ISA because the ILP is dynamically extracted by hardware. The out-of-order execution used in superscalar processors more aggressively extracts ILP by scheduling instructions regardless of the original program order.

To utilize hardware resources in a superscalar more effectively, simultaneous multi-threading (SMT) such as hyper threading is proposed. SMT is a technique which exploits thread-level parallelism (TLP); two or more threads are simultaneously executed on a superscalar. Threads share abundant execution units, such as arithmetic and logic unit (ALU) and load/store unit (LSU), and this enhances utilization of hardware resources.

Unfortunately, extracting more ILP in a single-thread has increased the complexity of hardware which is not reasonable for the performance gain.

2.2 Proliferation of Multi-core and Many-core Architectures

Difficulty of improving single-thread performance with a reasonable hardware results in a recent shift from single-core to multi-core design. This trend has aimed to increase the number of cores along with increase in available transistor count on a chip to improve computing performance. Processor architects have started focusing on multi-core and many-core architectures. A multi-core processor composed of the same type cores is referred to as homogeneous multi-core. Currently, many of commercial products employs the homogeneous structure, e.g., Intel core series, Oracle UltraSPARC series. With multiple cores on a chip, throughput of the processor increases because multiple applications can be simultaneously executed on the chip.

In the last decade, processor architects have increased the number of core for performance growth owing to innovative technology scaling, rather than focusing on single-core performance. Most of modern personal computers and high-end mobile computers have two or four processor cores on a chip. As for processors used in server computers, a larger number of processor core is generally employed, 16 cores of UltraSPARC T5, 60 cores of Xeon Phi.

However, the failure of Dennerd scaling imposes a wall on processor architects. Utilizable core count in parallel is limited due to power budget and heat density; consequently, a part of a chip is not simultaneously activated (the area is called "dark silicon"). The percentage of dark silicon will increase if processor architects increase core count like the last decade [27].

2.3 A New Paradigm: Heterogeneous Multi-core Architecture

Currently, energy efficiency becomes the forefront of the design constraints in processors due to the dark silicon problem described above. Designers have started improving energy efficiency at the circuit and microarchitectural levels. The power wall has already made a disruptive impact on the computing industry in the last decade - major processor companies shifted to multi-core based designs from conventional monolithic superscalar designs [28]. Furthermore, heterogeneous multi-core architectures start getting much attention [29, 30].

Heterogeneous multi-core approach is classified into two categories: single-ISA and multi-ISA. A single-ISA heterogeneous multi-core provides diverse superscalar core types, each core is customized to an intended behavior of program, program phase, or class of program behaviors. Using a proper superscalar core for characteristic in a program contributes to reduce energy consumption and yield higher performance on individual programs. For this reason, many researchers have become focusing on single-ISA heterogeneous multi-core processors. Companies have started adopting the architecture into their product, e.g., ARM's big.LITTLE processing [3] and NVIDIA's variable symmetric multiprocessing [31]. In contrast, a multi-ISA heterogeneous multi-core generally consists of one or more general purpose processor with digital signal processor (DSP) / graphic processing unit (GPU) / accelerator. Applications executed on a multi-ISA heterogeneous multi-core are programmed and compiled for target processing unit and assigned onto the processing unit; each application is effectively executed on targeted processing unit.

Single-ISA heterogeneous multi-core processor has advantages of appli-

cation portability and flexibility compared with multi-ISA heterogeneous multi-core processor. This is because executable binary files can be efficiently executed on succeeding processors of the same ISA, which may improve the microarchitecture, while multi-heterogeneous system often requires to recompile applications to be executed on or optimized for a new product because of the difference of available hardware resources (the number of functional units).

2.4 Necessary Requirements for Developing Low-energy Single-ISA Heterogeneous Multi-core Processors

Although single-ISA heterogeneous multi-core processor can achieve a higher-performance computing with lower-energy consumption, there are two challenges: (1) developing energy-efficient cores used in a heterogeneous multi-core, and (2) improving research productivity of heterogeneous multi-core processor.

(1) In modern processors including heterogeneous multi-core, energy reduction of each processor core is still in the spotlight. In particular, D-flip-flops have a large impact on the energy and performance of a processor. In order to develop low-energy heterogeneous multi-core, the author proposes two novel D-flip-flops in Chapter 3: one achieves higher performance to be used for high-end cores, and the other saves larger energy intended to use for low-end but low-energy cores in a heterogeneous multi-core processor, respectively.

Reducing further energy consumption on a heterogeneous multi-core requires detecting fluctuation of program behavior and migrating program to another core. However, since a migration takes considerable overhead, frequent migration is not effective for energy saving. For this reason, the author

proposes a low-energy processor architecture which is effective for optimizing fine-grained fluctuation of program behavior on a heterogeneous multi-core processor in Chapter 4.

(2) The other challenge in heterogeneous paradigm is to improve research and development productivity. Processor designers have to dedicate an enormous effort to design a heterogeneous multi-core processor because it requires diverse cores, caches, and a bus system on a chip. In the current situation, processor companies often reutilize their past products to design their heterogeneous multi-core product. One of the reasons of the lack of microarchitectural diversity on a chip is due to the design effort problem. To mitigate the design effort, the author proposes FabHetero which is a framework for automatically generating RTL design of heterogeneous multi-core processors in Chapter 5.

Moreover, RTL design is often not the end-point of researches; producing physical design is imperative process but heavy task especially for a small research group. The author proposes a co-simulation framework to facilitate processor development on standard ASIC flows in Chapter 6.

3 Low-energy D-flip-flop

This chapter proposes two novel D-flip-flops used to design energy efficient processor cores. Section 3.1 classifies D-flip-flops according to circuit schematics. Section 3.2 describes a conventional low-energy D-flip-flop and its problems. Section 3.3 details proposed D-flip-flops. Section 3.4 presents the evaluation results of the proposed D-flip-flops.

3.1 D-flip-flop Classification

This work focuses on the semi-static and true-single-phase clocking schemes because it can achieve high speed and low-power consumption on the conventional digital circuit design. This section describes the classification of D-flip-flops and provide schematics of the representative D-flip-flops.

Static D-flip-flop

The modified C^2MOS D-flip-flop shown in Fig. 3.2 is based on a static master-slave latches scheme, and this scheme is one of the most widely used circuits in digital circuit design due to its good robustness of operation. Static master-slave D-flip-flops have feedback circuits in both the master and slave latches. Therefore, static master-slave D-flip-flops can maintain their stored value even if the clock is stopped.

Dynamic D-flip-flop

The dynamic transmission gate D-flip-flop shown in Fig. 3.3 is based on the dynamic structure. Typically, dynamic D-flip-flops can achieve higher speed and lower-power consumption. However, in dynamic D-flip-flops such as that shown in Fig. 3.3, the stored value will be destroyed if it is not refreshed for

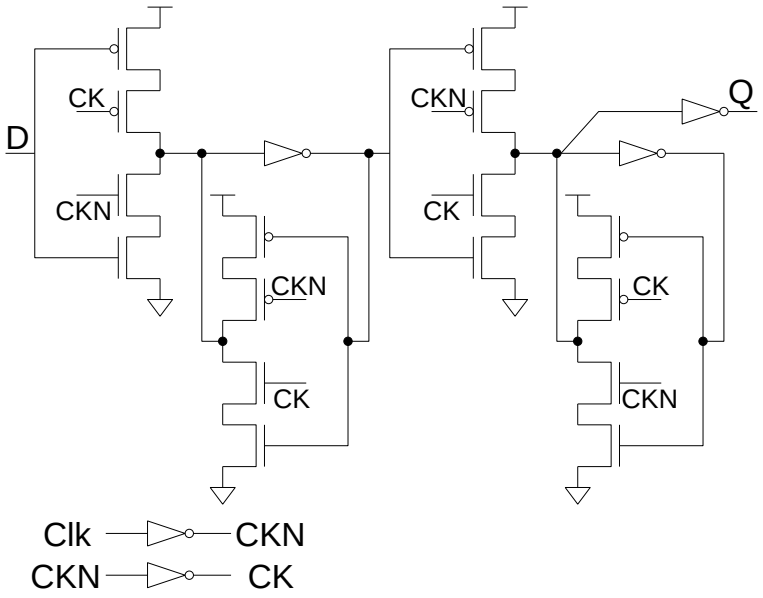

Figure 3.2: Schematic of modified C^2MOS D-flip-flop.

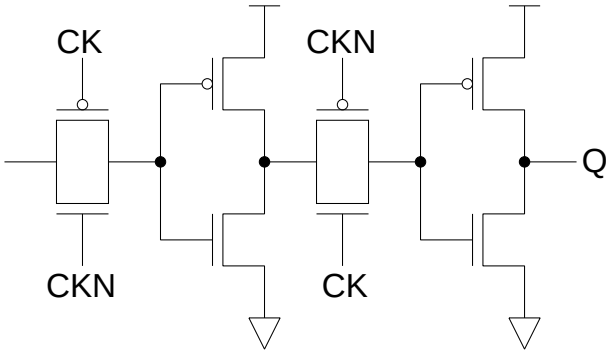

Figure 3.3: Schematic of dynamic transmission-gate D-flip-flop.

a long time. Therefore, designers must consider storage loss due to leakage current.

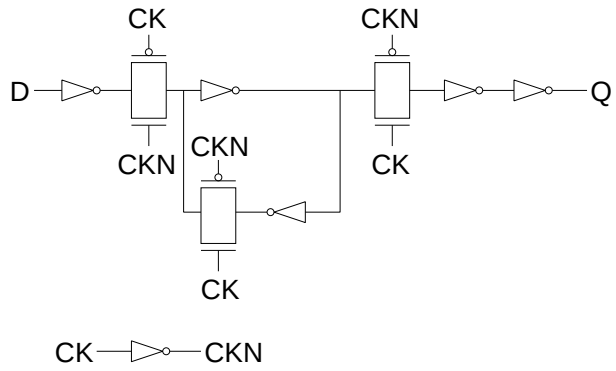

Figure 3.4: Schematic of semi-static flip-flop.

Semi-static D-flip-flop

In contrast, although semi-static D-flip-flops such as shown in Fig. 3.4 achieve an intermediate performance between static master-slave D-flip-flop and dynamic D-flip-flop, semi-static D-flip-flops can maintain their stored value when clock is ether high or low level. Consequently, the semi-static scheme solves the constraint of dynamic D-flip-flops.

Pulsed D-flip-flop

As conventional fastest D-flip-flops, pulsed flip-flop such as the Hybrid-Latch Flip-Flop (HLFF) [32] and Semi-Dynamic Flip-Flop (SDFF) [33] are proposed. Those D-flip-flops consist of two stages: the front-end stage functions as a pulse generator and the back-end stage captures input data (this action is triggered by the generated pulse). Although they achieve a high performance, their power consumption is typically not low. In order to reduce the delay, area, and power of the above pulsed flip-flops, Kumar et al. propose novel pulsed flip-flop [34]. To reduce the power consumption, the D-flip-flop based on split-output TSPC latch and the Pulse-Triggered TSPC Flip-Flop

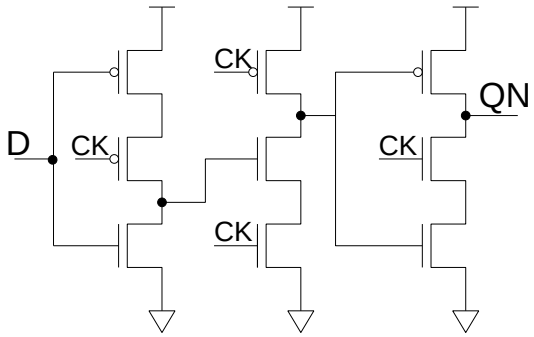

Figure 3.5: Schematic of basic TSPC D-flip-flop.

are also proposed [35]. In most pulsed flip-flops, a separate pulse generator and latches are used and the pulse generator must be shared by an adequate number of latches. Since this optimization requires manual effort, it is difficult to use automated digital circuit designs.

The author has targeted the conventional digital circuit designs; therefore, dynamic D-flip-flops and pulsed flip-flops are not in the scope of this work.

True-single-phase clocking (TSPC) D-flip-flop

The TSPC scheme has been demonstrated to be an efficient methodology to achieve high-speed and low-power VLSI design [36]. Fig. 3.5 shows a schematic of the basic TSPC D-flip-flop. Since the TSPC scheme can reduce the number of clock-driven transistors, it has the advantage of the power consumption of a clock system. However, it is difficult to use the basic TSPC D-flip-flop in standard cell based design because it is based on a dynamic structure. In addition, because there is a pre-charged node in the circuit, the power consumption of the clock system is large.

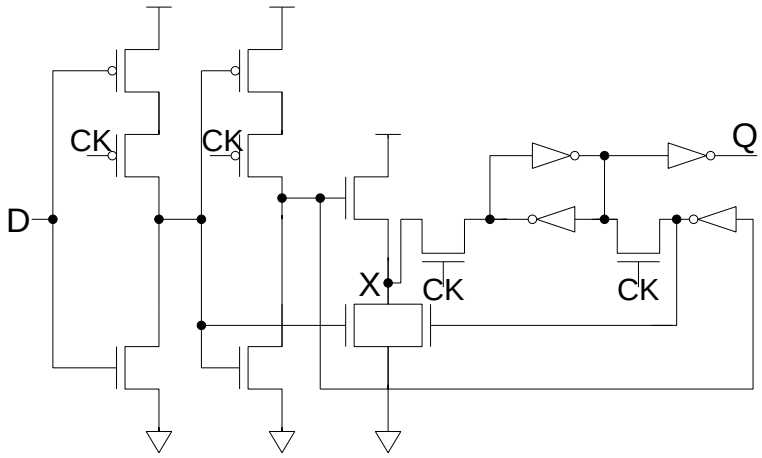

Figure 3.6: Schematic of HSTSPC D-flip-flop.

3.2 High-performance Semi-static TSPC D-flip-flop

In order to solve the problems of the basic TSPC D-flip-flop, high-performance semi-static TSPC (HSTSPC) D-flip-flop shown in Fig. 3.6 is proposed [5]. The master latch of the HSTSPC D-flip-flop is constructed with serial clocked inverters, called static P-stages, which are based on a dynamic structure. As the feature of the HSTSPC D-flip-flop, the nodes of the cross-coupled inverters, which are gated clock by two NMOS pass-transistors, are driven by two sides: one node is pulled up and the other is pulled down by two outputs of the master latch. For this reason, it is possible to speed up the write operation and reduce the power consumption caused by short-circuit current. The HSTSPC D-flip-flop has 5, 15% lower delay and consumes 70, 50% less power than the modified C^2MOS D-flip-flop in 180 nm, 90 nm technology, respectively. In addition, because the HSTSPC D-flip-flop employs semi-static structure, it is easy to replace conventional static master-slave D-flip-flops in the conventional digital circuit design.

3.2.1 Problems of High-performance Semi-static TSPC D-flip-flop

However, the HSTSPC D-flip-flop has two problems.

The number of NMOS and PMOS transistors in an HSTSPC D-flip-flop is 11 and 8, respectively, and this makes an unbalanced circuit. Consequently, a large dead space appears in layout design using the conventional layout model shown in Fig. 3.7(A). Fig. 3.7(B) shows the proposed layout model from Ref. [5] to solve this problem. However, it is difficult to optimize the layout area with a special layout model compared with a conventional layout model.

The second problem is slow-operation speed of the front-end. When input D in Fig. 3.6 is high level, path of D to X becomes the critical path and node X is pulled up by the NMOS transistor. For this reason, node X does not fully drive to high level, and this causes slowing down of the write operation.

3.3 Split-output Latch Based D-flip-flops

This section proposes D-flip-flops which improve HSTSPC D-flip-flop by solving the problems.

3.3.1 Double split-output semi-static TSPC D-flip-flop (DSSTSPC D-flip-flop)

Fig. 3.8 shows a schematic of the speed-efficient design, called double split-output semi-static TSPC (DSSTSPC) D-flip-flop. The features of the DSSTSPC D-flip-flop compared with the HSTSPC D-lip-flop are given as follows.

1. A pair of split-output latches [36] is used in parallel to produce normal and inverse outputs of the master latch. As a result, the critical path (D to X) is reduced to 2-stage, compared with the 3-stage of the HSTSPC

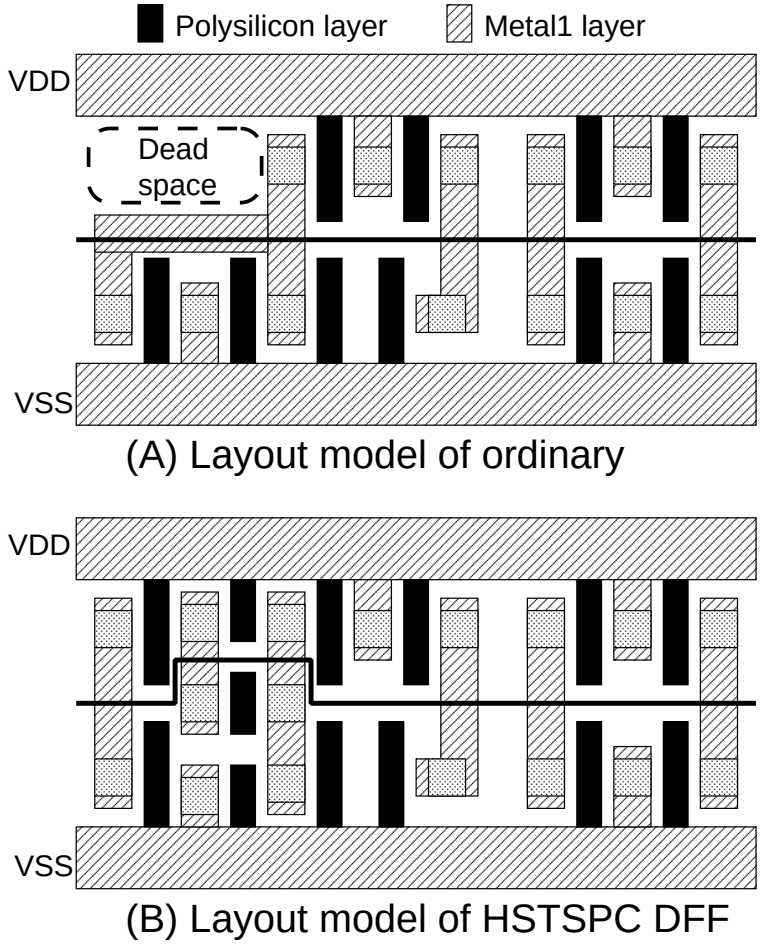

Figure 3.7: Layout model of the ordinary and HSTSPC D-flip-flop.

D-flip-flop. In addition, node X is pulled up using the PMOS transistor, so that the node X can be fully driven to high level.

2. The numbers of NMOS and PMOS transistors are the same. Therefore, the layout area can be more easily minimized than the HSTSPC D-flip-flop.

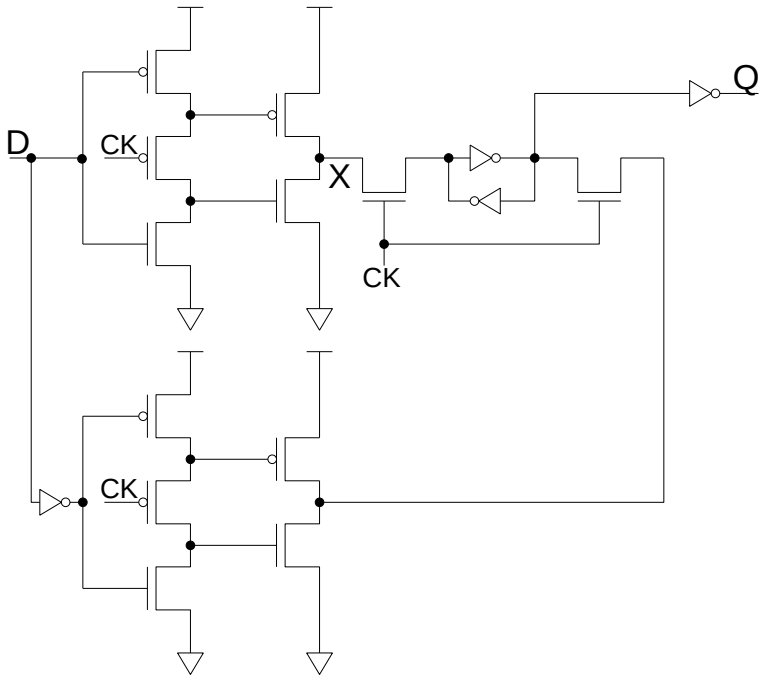

Figure 3.8: Schematic of DSSTSPC D-flip-flop.

These advantageous features mean that the DSSTSPC D-flip-flop can achieve less delay and smaller layout area than the HSTSPC D-flip-flop.

3.3.2 Single split-output semi-static TSPC D-flip-flop (SSSTSPC D-flip-flop)

Fig. 3.9 shows a schematic diagram of the SSSTSPC D-flip-flop which is the power and area-efficient design. In contrast to the HSTSPC D-flip-flop, the DSSTSPC D-flip-flop generates pull up and pull down outputs into cross-coupled inverters by using independent split-output latches. Therefore, by removing a split-output latch, which generates an inverse output of the master latch, the number of transistors can be reduced. However, by simply removing the split-output latch, the delay is increased significantly. In or-

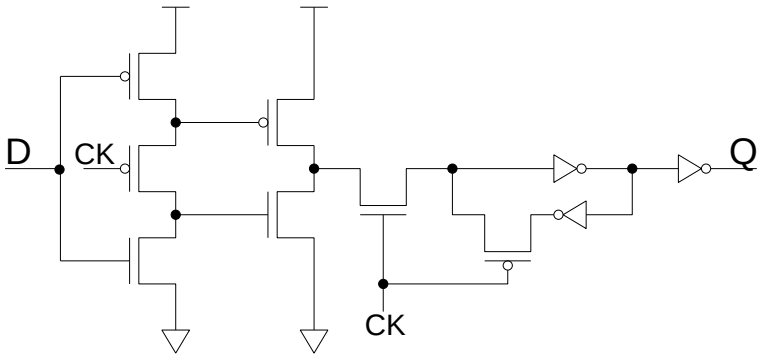

Figure 3.9: Schematic of SSSTSPC D-flip-flop.

der to reduce the delay, a clocked PMOS pass-transistor is inserted into the cross-coupled inverters, as shown in Fig. 3.9. Compared with the HSTSPC D-flip-flop, the SSSTSPC D-flip-flop can reduce the number of transistors by six that includes one clocked transistor consuming a large power.

Similar to the SSSTSPC D-flip-flop, it is possible to remove a part of the HSTSPC D-flip-flop. However, this schematic cannot reduce the clocked transistors, and there still remains unbalanced scheme in the number of NMOS and PMOS transistors. Therefore, the SSSTSPC D-flip-flop can reduce power consumption further and the layout area can be easily minimized.

3.4 Evaluation Results

The author performed SPICE simulations to evaluate the proposed D-flip-flops using net-lists extracted from physical design for Rohm 180 nm technology and Synopsys HSPICE. Therefore, the effectiveness of diffusion capacitance is considered. The performance of the proposed D-flip-flops are compared with the modified C^2MOS D-flip-flop, representative for the current digital circuit design, and HSTSPC D-flip-flop. In this section, all simulations are performed at typical situation: 1.8 V and 27 °C. In order to

obtain accurate results, the inputs of D-flip-flops are driven by input driver and the outputs are required to drive a load of ten inverters. Because total input capacitance of clock nodes in each D-flip-flop is different, the author used ten modified C^{2}MOS D-flip-flop, four HSTSPC D-flip-flop, four DSSTSPC D-flip-flop and six SSSTSPC D-flip-flops per clock driver. On the other hand, although input capacitance of the D-input of the DSSTSPC D-flip-flop is larger than the other D-flip-flops, each D-flip-flop are driven by the same input driver. Because the path that connects to D-input becomes a critical path of a sequential circuit, the node that connects to a D-input is not typically shared with other logic cells. The author examined all 5,577 D-flip-flops in a MIPS R3000 compatible processor, which has 9-stage pipeline, and found that each node that connects to a D-input of a D-flip-flop drives only one D-flip-flop. The evaluation results for the power consumption, delay and layout area are described in the following sections.

In addition, the author also evaluated in a PTM [37] 90nm technology to present the effectiveness on the finer process which is more impacted by leakage power.

3.4.1 Energy Consumption

Generally, the toggle rate of input data for a D-flip-flop is low; therefore, the power consumption is shown in Fig. 3.10 when the input toggle rate is 1 to 20% at 500 MHz for Rohm 180 nm. Although the data is not shown here, the power consumption of each D-flip-flops is linearly increased when the toggle rate is increased up to 100%. According to Fig. 3.10, the power consumption of the DSSTSPC D-flip-flop is 70% lower than the modified C^{2}MOS D-flip-flop when the toggle rate is around 5% (the percentage is said to be near the actual toggle rate [5]), and almost the same as that of the HSTSPC D-flip-

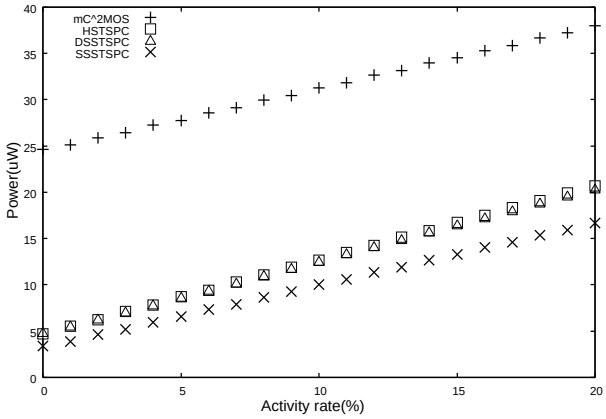

Figure 3.10: Power across different input toggle rate (180 nm).

flop. The SSSTSPC D-flip-flop can reduce the power consumption by 25% compared to the HSTSPC D-flip-flop when the toggle rate is around 5%.

3.4.2 Circuit Delay

Because the sum of the setup time (T_{su}) and clock to Q delay (T_{cq}) affects the clock frequency, the delay is evaluated using $T_{su} + T_{cq}$. Firstly, the author explains why $T_{su} + T_{cq}$ affects the clock frequency. $DELAY$ denotes the maximum delay of the combinational logic between the two D-flip-flops. If T is the clock period, then the expression $T > T_{su} + T_{cq} + DELAY$ must be met for correct operation. $DELAY$ depends on the combinational logic; therefore, the performance of a D-flip-flop is $T_{su} + T_{cq}$. If this quantity is lower, then the clock frequency will be higher and vice-versa.

Table 3.1 shows the evaluation results. The delay of the DSSTSPC D-flip-flop is reduced by 11% and 5% compared with the modified C^{2}MOS D-flip-flop and HSTSPC D-flip-flop, respectively. The delay of the SSSTSPC D-flip-flop is the same as that of the HSTSPC D-flip-flop. The transition of node X in the DSSTSPC D-flip-flop (Fig. 3.8) is faster than that of HSTSPC

Table 3.1: Comparison of delay (180 nm).

	T_{su} [ns]	T_{cq} [ns]	$T_{su} + Tcq$ [ns]
`modified C²MOS D-flip-flop`	0.21	0.58	0.79
`HSTSPC D-flip-flop`	0.29	0.45	0.74
`DSSTSPC D-flip-flop`	0.25	0.45	0.70
`SSSTSPC D-flip-flop`	0.25	0.49	0.74

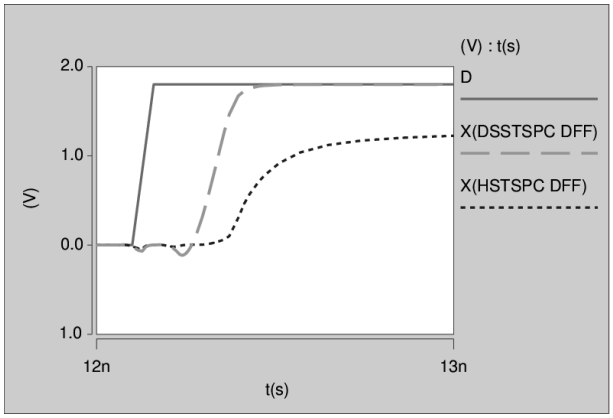

Figure 3.11: Transition of the node X (180 nm).

D-flip-flop (Fig. 3.6), so that the DSSTSPC D-flip-flop can reduce the delay. Fig. 3.11 shows the transitions for the node X of the HSTSPC D-flip-flop and DSSTSPC D-flip-flop. According to Fig. 3.11, node X of the DSSTSPC D-flip-flop drives to high level fully and faster than the HSTSPC D-flip-flop.

3.4.3 Layout Area

The author designed the layouts of the HSTSPC D-flip-flop, DSSTSPC D-flip-flop and SSSTSPC D-flip-flop on the Rohm 180 nm technology. The modified C²MOS D-flip-flop, HSTSPC D-flip-flop and DSSTSPC D-flip-flop occupy an area of $51.6\mu m^2$. Note that the HSTSPC D-flip-flop layout uses the special layout model shown in Fig. 3.7, and in conventional layout model,

it occupies an area of $64.5\mu m^2$. The SSSTSPC D-flip-flop occupies an area of $35.5\mu m^2$. The DSSTSPC D-flip-flop can reduce the layout area by 20% than that of the HSTSPC D-flip-flop using the conventional layout model, and is still the same as that with special layout model. Compared with the HSTSPC D-flip-flop with special layout model, the SSSTSPC D-flip-flop has 31% smaller layout area.

3.4.4 Case Study on 90 nm Technology

This section presents the evaluation result in the PTM 90 nm technology to demonstrate the effectiveness of the proposed D-flip-flops on the finer process. In this section, simulations are performed at 1.2 V and 27 °C.

Fig. 3.12 shows the power consumption evaluated in the same way as previous simulation (see section 3.4.1). The DSSTSPC D-flip-flop reduces the power consumption by 50% than the modified C^2MOS D-flip-flop and almost the same as that of the HSTSPC D-flip-flop. The SSSTSPC D-flip-flop consumes 30% lower power than the HSTSPC D-flip-flop when the toggle rate is around 5%. The leakage power is approximately 8% of the power of the DSSTSPC D-flip-flop, and it is obvious that proposed D-flip-flops can achieve low-power on the 90 nm technology.

Table 3.2 shows the result of the delay evaluation. The DSSTSPC D-flip-flop has 20% and 4% lower delay compared with the modified C^2MOS D-flip-flop and HSTSPC D-flip-flop respectively. Although the SSSTSPC D-flip-flop is 8% slower than the HSTSPC D-flip-flop, it is still 10% faster than the modified C^2MOS D-flip-flop.

3.4.5 Voltage tolerance and verification of semi-static structure

In order to reduce power consumption, dynamic voltage and frequency scaling (DVFS), which dynamically lowers the supply voltage and frequency, is

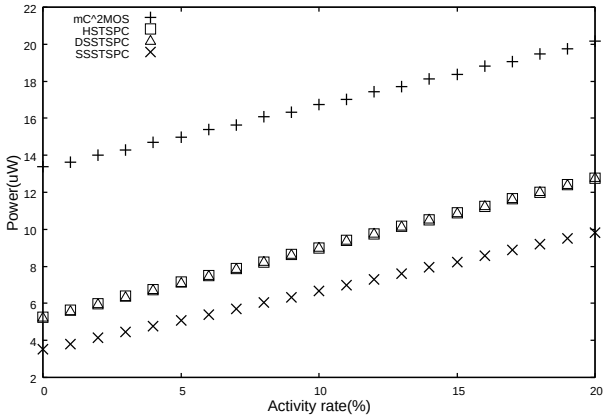

Figure 3.12: Power across different input toggle rate (90 nm).

Table 3.2: Comparison of delay (90 nm).

	T_{su} [ps]	T_{cq} [ps]	$T_{su} + Tcq$ [ps]
`modified C2MOS D-flip-flop`	61	215	276
`HSTSPC D-flip-flop`	64	165	229
`DSSTSPC D-flip-flop`	50	169	219
`SSSTSPC D-flip-flop`	50	197	247

currently employed in many applications. Therefore, the performance when lowering the supply voltage is very important. Fig. 3.13 shows relation between supply voltage and speed when the clock frequency is 1GHz. According to the result, the DSSTSPC D-flip-flop can operate more quickly than the modified C^{2}MOS D-flip-flop in the lower supply voltage. When the supply voltage is higher than 0.9 V, the SSSTSPC D-flip-flop is the same or better performance than the modified C^{2}MOS D-flip-flop. Because the DVFS technique requires control system (in most cases that is operating system) and lowering the supply voltage needs to be careful, the DVFS technique cannot be employed always. Consequently, the SSSTSPC D-flip-flop effectively

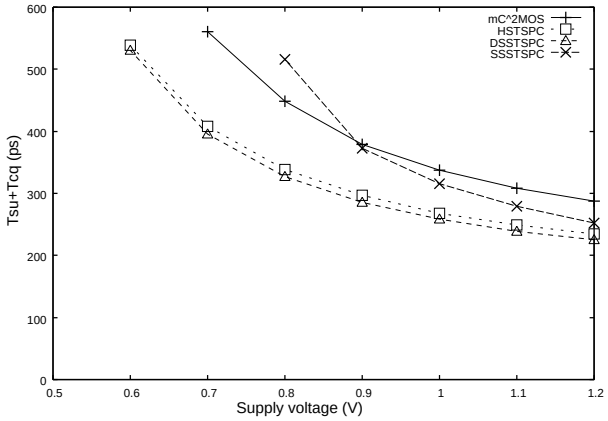

Figure 3.13: Relation between supply voltage and speed (90nm).

works in the embedded systems which aim at low power but do not apply the DVFS technique.

Proposed D-flip-flops can keep their stored data when clock is low level. However, they lose the data when clock keeps high level for a long time. Therefore, if the clock frequency is very slow, proposed D-flip-flops may not rightly operate. However, the author finds that proposed D-flip-flops can operate at 1MHz without increasing power caused by short-circuit current, which is due to use the semi-static structure. This means that proposed D-flip-flops are effective in the most digital circuits.

3.4.6 Summary of Evaluation Results

From those experiments, although the DSSTSPC D-flip-flop has 1 more transistor than that of the HSTSPC D-flip-flop, it has approximately 5% lower delay without disadvantage on both the 180nm and 90nm technology. Furthermore, although the speed of the SSSTSPC D-flip-flop is 8% slower than that of the HSTSPC D-flip-flop on the 90 nm technology, it the power consumpiton is reduced by more than 25% on both the 180 nm and 90 nm

technology. In addition, the layout area of the SSSTSPC D-flip-flop is 31% smaller than the other D-flip-flops.

4 Variable Stages Pipeline Architecture

This chapter describes efficient usage method and enhancement of VSP processor. Section 4.1 discusses a commonly used low-energy technique called DVFS and its problem. Section 4.2 introduces variable-depth pipeline architectures including VSP. Section 4.3 presents pipeline depth control method for variable-depth pipeline architectures. Section 4.4 details the implementation of VSP processor that the author fabricated and used to evaluation in this thesis. Section 4.5 presents evaluation results of VSP and its control method. Section 4.6 and 4.7 discuss glitch propagation problem occurred on variable-depth architectures and a solution, called LDS-cell. Section 4.8 presents the effectiveness of LDS-cell. Section 4.9 summarizes this chapter.

4.1 Dynamic Voltage and Frequency Scaling

The latest advances in mobile computers, such as personal digital assistants and smart phones, have led to a goal of higher computing performance with lower energy consumption. Dynamic voltage and frequency scaling (DVFS) [8, 9] which dynamically lowers the supply voltage and clock frequency is currently used in many processors to reduce energy consumption. Lowering the supply voltage effectively reduces energy consumption because energy consumption depends on the square of supply voltage.

However, it is difficult to deliver fine-grained energy reduction using DVFS technique because voltage scaling takes a long time and charging/discharging a power supply line consumes a large energy consumption. Therefore, the useful interval of DVFS is limited to coarse-grain. Reference [10] estimates the useful interval of DVFS in terms of energy reduction and it takes at least 10^5 cycle order to reduce energy consumption. In general, it is said that monitoring the workload of a processor and scaling supply voltage take tens

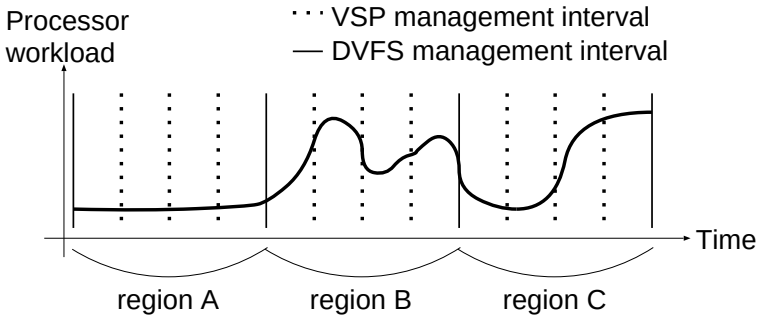

Figure 4.14: Relation between workload and energy management interval.

or hundreds of microseconds. This is shown in Fig. 4.14 for the lines labeled "DVFS management interval". In region A, DVFS can effectively reduce energy by lowering the voltage and frequency because the workload is light through the DVFS management interval. In region B and C, however, the workload transitions at shorter interval than the DVFS management interval. If DVFS raises the voltage and frequency for the peak workload in region B, the processor wastes energy at low workload region in region B. In contrast, if DVFS sets lower voltage and frequency than that of the peak, response time is degraded.

In addition, the author points that the effectiveness of DVFS is limited because there is a lower bound on voltage of minimum operating voltage of CMOS devices. Figure 4.15 shows this problem. A processor that only DVFS is applied reduces energy consumption by lowering the voltage and frequency as shown point A to B in Fig. 4.15. However, once the voltage has reached the lowest voltage, lowering the frequency does not save further energy consumption as shown point B to C because lowering the frequency without lowering the voltage reduces only power. Therefore, developing low-energy techniques that compensate for DVFS is essential.

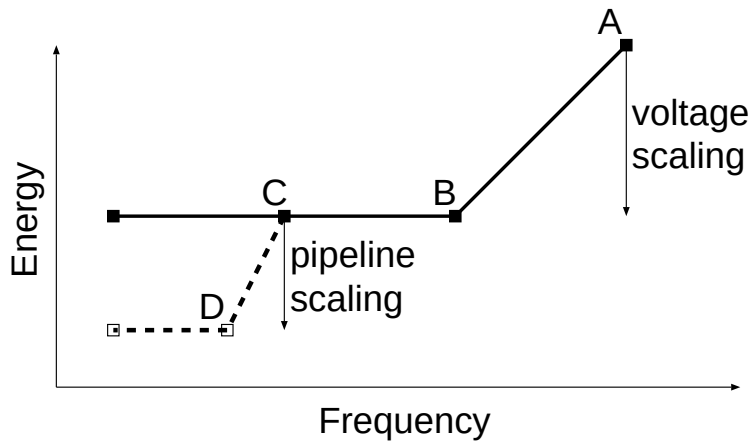

Figure 4.15: Combining voltage scaling and pipeline scaling.

4.2 Variable-depth Pipeline Architectures

As one of architectural low-energy technique, variable-depth pipeline architectures are proposed; Shimada, et.al. propose pipeline stage unification (PSU) [42–45], Koppanalil, et.al. propose dynamic pipeline scaling (DPS) [46] and we propose VSP [19, 39–41]. Figure 4.16 shows the basic concept of these approaches. Generally, if the pipeline depth is increased, such as in super-pipeline architecture, then the clock frequency, performance, and energy consumption are also increased. In contrast, if the pipeline depth is decreased, then the clock frequency, performance, and energy consumption are also decreased. When the processor workload is light, the processor lowers clock frequency and unifies plural stages to form a shallower pipeline.

Figure 4.17 shows the pipeline register used in PSU processor that can vary the depth of the pipeline stages dynamically. Under high-speed mode, the D-flip-flop+MUX shown in Fig. 4.17 acts as a general pipeline register by selecting an upper path. Under low-energy mode, the D-flip-flop+MUX unifies the pipeline stages by selecting a lower path to connect the input port

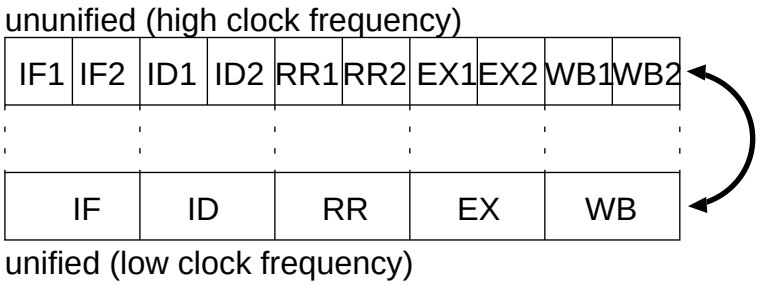

Figure 4.16: Variable-depth pipeline architecture.

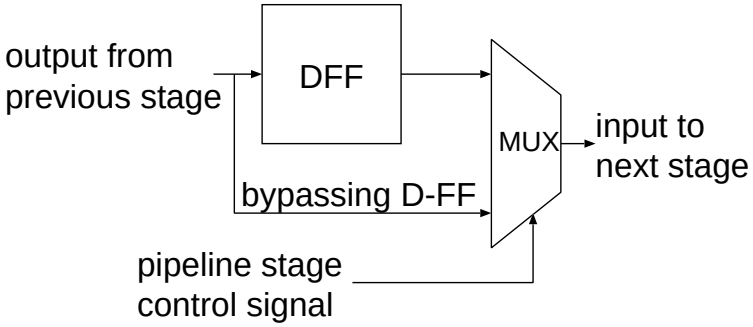

Figure 4.17: Circuit diagram of pipeline registers to vary pipeline.

to the output port.

Unlike DVFS, these architectures can save energy consumption in an interval of several ten or hundred clock cycles because the penalty of varying the pipeline depth takes only a few clock cycles. Therefore, fine-grained energy optimization is possible if variable-depth pipeline architectures change the pipeline depth to a suitable pipeline depth using a fine-grain controller. The advantage of fine-grained energy optimization is shown in Fig. 4.14 for the doted lines labeled "VSP management interval". In region B and C, more aggressive energy reduction is performed by using VSP which has fine-grain controller. Furthermore, even if the workload requires middle performance of

the processor, VSP can reduce energy consumption like DVFS but the useful interval is finer-grained. To reduce more energy consumption, VSP can be used with DVFS at the same time. The point D in Fig 4.15 shows that VSP can reduce energy consumption beyond the lower bound of DVFS. Ref. [46] points this and briefly estimates the effectiveness.

4.3 Fine-grain Depth-change Controller

To reduce energy consumption with minimum performance degradation, it is essential to correctly change the pipeline depth to a suitable pipeline depth. Therefore, we propose a fine-grain pipeline depth controller shown in Fig. 4.18 that stores some processor states (correlating to the workload of the processor) in the latest several tens of cycles and predicts a suitable pipeline depth using the stored information. In this approach, we can choose information such as cache hit ratio, the number of cache access, instruction per cycle (IPC), and so on, to predict a suitable pipeline depth.

The author uses the instruction per cycle (IPC) as the parameter shown in Fig. 4.18 to predict suitable pipeline depth. IPC is a good parameter for estimating processor workload and program's instruction-level behavior. Under high-speed mode, if the IPC is larger than threshold, the VSP processor remains unchanged because the pipeline effectively works. On the other hand, if the IPC is lowered, the VSP processor shifts to low-energy mode. Under low-energy mode, if the IPC is larger than threshold, then the VSP processor shifts from low-energy mode to high-speed mode, and vice versa. However, a problem occurs when the processor shifts from low-energy mode to high-speed mode. In general, a large number of branch mis-predictions degrades the IPC but this IPC degradation does not occur in low-energy mode. If the VSP processor runs under low-energy mode in

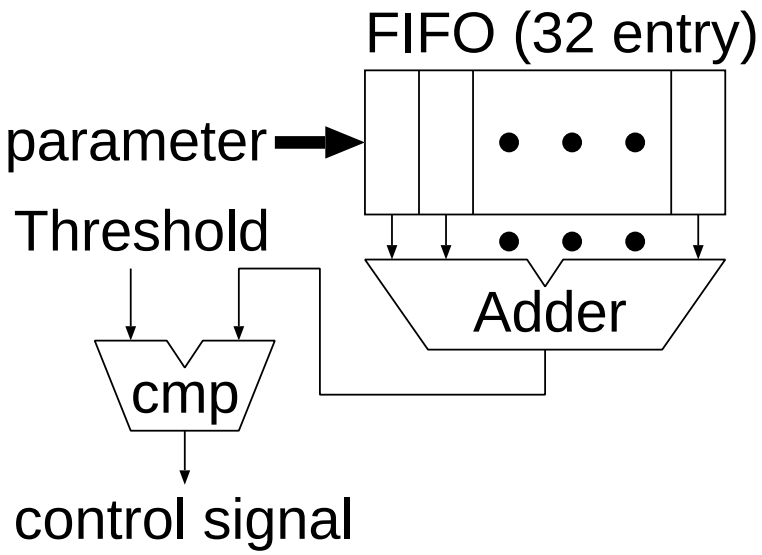

Figure 4.18: Block diagram of fine grain controller.

branch mis-prediction intensive phase, the IPC approaches the highest IPC because branch mis-prediction does not occur due to pipeline unification and then the VSP processor shifts back to high-speed mode. As a result, the VSP processor runs an improper pipeline depth and the performance degrades. To solve the problem, the author also uses the number of branch instructions to predict when the processor shifts to high-speed mode because a branch mis-prediction intensive phase includes a large number of branch instructions. Therefore, three thresholds are used: IPC_{HtoL} for unifying pipeline stages, IPC_{LtoH} and $\#BR$ for forming deeper pipeline. These thresholds can be set by software: writing a new value to co-processor register unused by the base processor. An appropriate thresholds configuration according to workload is described in Section 4.5.

4.4 Implementation

This section describes the specific processor architecture used in this work.

4.4.1 Processor Architecture

A MIPS R3000 compatible processor, which is seven stages in-order processor, was used. The processor takes one cycle for branch instructions, two cycles for most integer ALU instructions and four cycles for complex instructions (multiplication and division). Generally MIPS processors do not adopt ALU of plural stages, however we employed such implementation to balance cycle time in each pipeline stage. As for our pipeline balancing, we hypothesize that cache system uses a high speed memory and/or is also pipelined like modern processors hence a path through instruction and data cache is not the critical path. We divided the ALU and MDU to two and four stages respectively in order to adjust the cycle time to around 5.9 ns because the longest stage except for the EX stage takes 5.9 ns. After the pipelining, the cycle times of pipelined ALU and MDU are 5.1 ns and 6.4 ns, respectively. The processor has 1 K-entry gshare branch predictor. Figure 4.19 shows the pipeline stages of the VSP processor. We call deeper (7-stage) pipeline *high-speed mode* and shallower (3-stage) pipeline *low-energy mode*. The features of low-energy mode are the following:

- The branch predictor is stopped because branch mis-prediction does not occur.
- No interlock occurs by data dependency between arithmetical operations.
- Clock lines for unused pipeline registers between unified stages are gated.

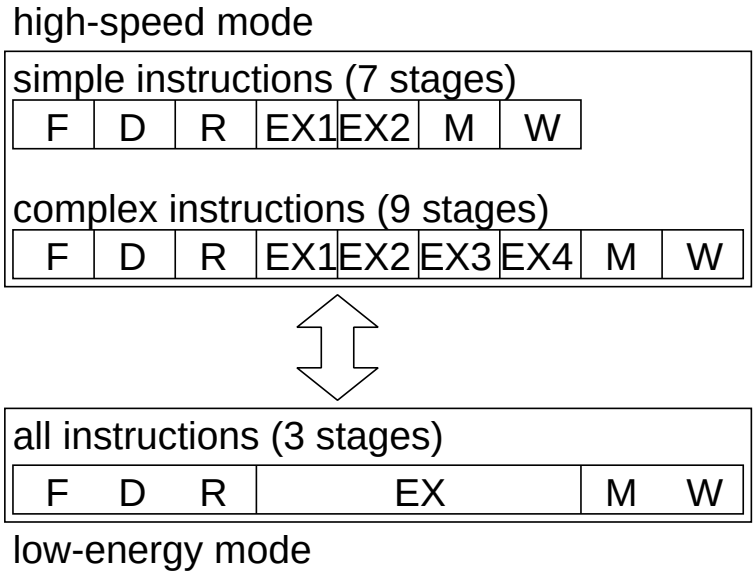

Figure 4.19: Block diagram of R3000 based VSP processor.

4.4.2 Low-overhead Depth Changing Technique

With proposed controller, the pipeline depth is always suitable for a running program. However, if the pipeline depth frequently changes, execution time will be increased. Forming a deeper pipeline does not have any problem. We assume that all pipeline stages execute valid instructions under low-energy mode and the processor makes deeper pipeline. In this case, to execute a program correctly, the VSP processor only has to perform the following step. The F, EX1 and M stages execute next instructions and the other stages just purge the holding instructions.

In contrast, when unifying the stages, the processor requires a pipeline flush because following instruction destroys preceding valid instructions. For example, the F, D, and R stages in Fig. 4.20(A) are executing valid instructions and if they are immediately unified, the instructions in the D and R

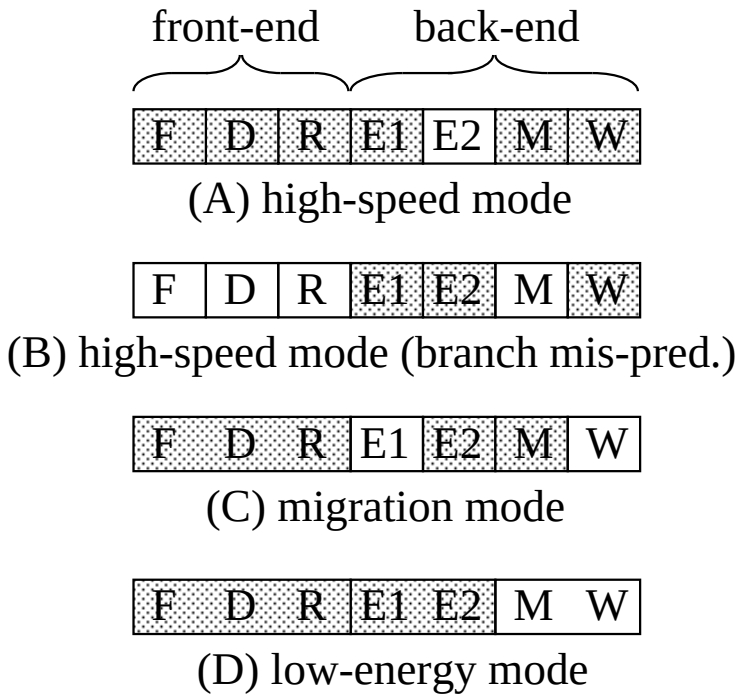

Figure 4.20: Change pipeline depth through migration mode.

stage are destroyed by the instruction in the F stage. As a result, the processor goes into an unrecoverable state. If the processor flushes the pipeline and restarts the instructions in order to solve the problem, seven clock cycles are needed for an unification.

Therefore, we hide the penalty by overlapping a pipeline flush caused by a branch mis-prediction. Furthermore, to finely control the high-speed and low-energy modes, we divide the pipeline stages into two groups: the front-end group contains instruction fetch, instruction decode and register read stages, and the back-end group contains the latter stages. We assume that the processor runs under high-speed mode as shown in Fig. 4.20(A), and valid instructions exist in the shaded boxes (F, D, R, E1, M and W). A branch mis-prediction occurs at the branch instruction in the E1 stage. We note

that the instruction in the R stage is a branch delay slot which must not be purged in a pipeline flush caused by a branch mis-prediction. When a branch mis-prediction occurs and pipeline depth controller is asserting pipeline unification signal, instructions in the front-end stages are purged and the branch delay slot enters the E1 stage as shown in Fig. 4.20(B). In the next cycle, the processor shifts to migration mode as shown in Fig. 4.20(C) and a new instruction is fetched. Under migration mode, the front-end stages are unified and driven at low clock frequency (in this case quarter of high-speed mode), and the back-end stages are driven at high clock frequency. It takes four high clock frequency cycles that the new fetched instruction reaches the E1 stage. Therefore, instructions in the back-end stages can be retired except for rare case.[1]. When all the back-end stages become empty, they shift to low-energy mode as shown in Fig. 4.20(D).

In this way, the controller can hide the penalty of changing the pipeline depth. Generally, a branch mis-prediction occurs with a probability of several percent, and a branch instruction exists with a probability of 20%. Therefore, there are sufficient opportunities to change the pipeline depth in tens or hundreds of cycles.

This technique obtains more effectiveness in a superscalar and super-pipeline processor which have a large pipeline flushing penalty. The effectiveness of this technique is discussed in section 4.5.

4.4.3 Chip Fabrication

The author implemented the VSP processor described the previous section in Verilog HDL. Table 4.3 shows the EDA tools used for designing VSP chip. Rohm 180 nm technology and Kyoto University standard cell library [47]

[1]If the branch delay slot is complex instruction that takes multi cycles in the execution stage

Table 4.3: EDA environment for fabricating chip

Phase	EDA tool
functional verification	Synopsys VCS, vers. 2009.06
synthesis	Synopsys Design Compiler, vers. 2010.03-SP5
place & route	Synopsys Astro, vers. 2007.03-SP12
LDS-cell design	Cadence IC, vers. 5141
design rule check	Mentor Graphics Calibre, vers. 2010.4

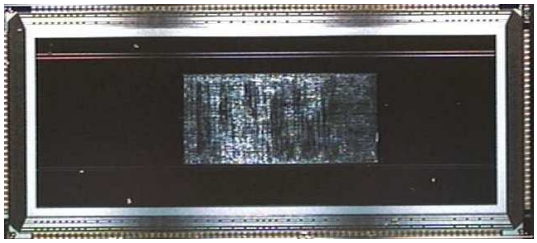

Figure 4.21: Chip micrograph.

except for LDS-cell are used for chip fabrication. The processor was designed to operate at 100MHz for high-speed mode and 25MHz for low-energy mode.

A cache system was not implemented in the chip to evaluate an accurate energy consumed by the VSP processor. However, this implementation does not degrade the accuracy of evaluation because in an in-order processor, the energy consumption caused by a cache miss under high-speed and low-energy modes are the same if the processor is applied clock gating when a cache miss occurs.

Figure 4.21 shows a micrograph of the fabricated VSP chip. The VSP processor chip contains 492,742 transistors.

4.5 Evaluations

4.5.1 Evaluation Methodology

Four integer benchmarks (bit count, int sqrt, quick sort and string search) from MiBench benchmark suit [48] were used for the evaluation. To clarify short-period energy reduction, the author adjusted these benchmarks to finish approximately 100,000 clock cycles. Evaluations of the energy were measured by using the fabricated chip. The author confirmed that the fabricated chip executes the benchmarks successfully. Although the author designed the chip to operate at 100 MHz for high-speed mode and 25 MHz for low-energy mode, the fabricated chip was evaluated at 35 MHz and 8.75 MHz respectively because of limitation of test environment.

4.5.2 Energy Consumption

Figure 4.22 compares energy consumption and execution time on diverse combinations of thresholds. HS mode and LE mode of the horizontal axis show results when the processor runs at fixed (high-speed/low-energy) mode and the others (TH1 to TH4) represent different combinations of three thresholds which are configured to achieve lower energy to the right. The vertical axis is normalized to result of high-speed mode. In the LE mode, the VSP processor reduces energy consumption by 34% to 48% compared with the HS mode. The energy advantage is due to higher IPC and clock gating for unused pipeline registers and branch predictor. TH1 to TH4 interpolate energy consumption and performance between high-speed and low-energy modes. This means the VSP reduces energy consumption at a short-interval (less than 100,000 cycles) according to configured thresholds even if the processor is required to achieve middle performance. In other words, the VSP can optimize trade off between the performance and energy in the region B and

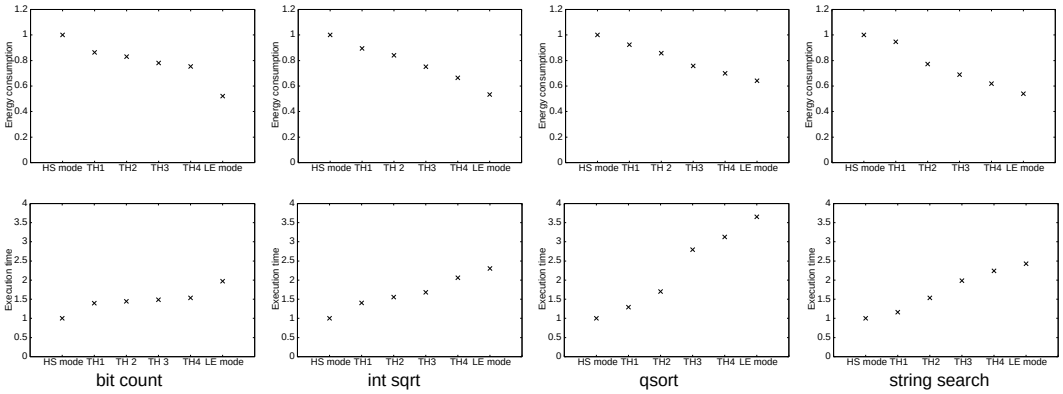

Figure 4.22: Energy consumption and execution time on diverse combination of thresholds.

C shown in Fig. 4.14. Dynamic thresholds configuration to produce more flexible optimization is left for future work, but section 4.5.5 presents case study for threshold optimization.

Figure 4.23 shows the effectiveness of combining DVFS and VSP. The author used 1.44 V (20% lower than standard voltage of the process) for the lowered voltage which is calculated by the margin for 45 nm Intel Atom processor [49]. The DVFS on rigid pipeline consumes approximately 62% energy of the rigid pipeline with standard voltage. This result is consistent with that energy depends on the square of voltage. The vertical axis of Fig. 4.23 is normalized to the result of DVFS on rigid pipeline to make the effectiveness of combining two techniques clear. Although the VSP reduces larger energy than the DVFS except for qsort, note that DVFS might be more energy efficient than VSP on a leakage dominant process because DVFS also is effective for leakage current. The DVFS on VSP reduces energy consumption by 38% to 52% with respect to the DVFS on rigid pipeline. It is obvious that using DVFS and VSP at the same time significantly reduces energy consumption as shown in Fig 4.15.

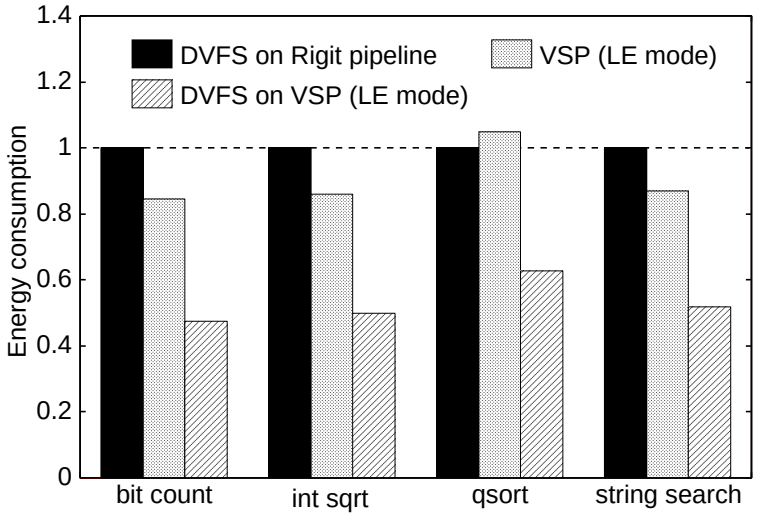

Figure 4.23: Measurement result of combining DVFS and VSP.

Finally, energy consumed by pipeline depth controller was measured. The fabricated chip has an input to disable (apply clock gating for) the pipeline depth controller. When the controller is disabled, the VSP runs as fixed high-speed mode or fixed low-energy mode (the mode is specified externally). The author compared controller-disabled VSP which runs as fixed high-speed mode with controller-enabled VSP which is configured its thresholds to run as fixed high-speed mode. As a result, it makes clear that the controller consumes only 2.1% energy of the VSP. This means the hardware cost of the controller is small enough.

4.5.3 Effectiveness of Low-overhead Depth Changing Technique

The author briefly estimated the effectiveness of low-overhead depth changing technique described in section 4.4. Table 4.4 shows performance degradation caused by pipeline unification in case of without the low-overhead technique. The author assumed a pipeline flush takes 7 cycles and this penalty is required

Table 4.4: Estimation for overhead caused by pipeline unification.

benchmark	combination of thresholds	average interval between unifications	performance degradation
bit count	TH1	892 cycle	0.8%
int sqrt	TH3	156 cycle	4.5%
qsort	TH1	219 cycle	3.2%
string search	TH4	142 cycle	4.9%

when the VSP shifts from high-speed mode to low-energy mode. The second column of the table shows combinations of thresholds shown in Fig. 4.22 which have the worst performance degradation in the benchmark and the third column denotes averaged interval between unifications (only shifting from high-speed mode to low-energy mode). Therefore, additional 7 cycles are required every averaged interval and this increases execution time by 0.8% to 4.9%.

In contrast, low-overhead technique hides the penalty. The author confirmed that migration mode always operates for 4 high-frequency cycles, i.e. the new fetched instruction in FDR stage shown in Fig. 4.20 is not stalled. Therefore, the VSP unifies the pipeline stages with no penalty.

4.5.4 Hardware Cost

Although the author did not fabricate the base processor chip that does not apply a variable-depth pipeline architecture, the author performed physical synthesis on the base processor to evaluate the additional hardware cost and energy of the VSP processor. The base processor has 456,022 transistors and VSP processor has 492,742 transistors. Therefore, modification for the VSP needs additional 8% hardware (36,720 transistors). The breakdown of the additional hardware is the following. Approximately 2800 multiplexers were

added to implement VSP, therefore,
$2,800\,multiplexers \times 12\,transistors = 33,600\,transistors$
are added. In addition, the depth changing controller increases 4705 transistors.

The author compared energy consumption of the VSP with that of the base processor by using Synopsys Nanosim which is a fast-SPICE circuit simulator. According to the simulation result, the VSP consumes 1.5% to 5% larger energy than the base processor. These cost are reasonable compared with obtained effectiveness.

4.5.5 Case Study for Sophisticated Energy Optimization

This section discusses how to decide appropriate thresholds to the VSP processor. To reveal relation between combination of thresholds, execution time and energy consumption, the author analyzed execution results on 88 *10-million SimPoints* [50] of four SPEC 2000 integer benchmarks (gzip, mcf, parser and bzip). Since used evaluation board can execute a program up to one million cycles, the author used a MIPS R3000 simulator to perform fast-forwarding and to create a checkpoint. The VSP chip restores processor state from the checkpoint and starts execution for measuring. In this way, the first 500,000 instructions of each SimPoint were executed in order to evaluate execution time and energy consumption. According to analysis, all 88 SimPoints are classified as two categories. Figure 4.24 and 4.25 show major SimPoints and minor SimPoints, respectively. In Fig. 4.24 and 4.25, each data indicates a SimPoint in a benchmark program (e.g., gzip_235 means the SimPoint which starts gzip program after skipping $235 \times 10\,million$ instructions). In these figures, TH0, TH1, TH2 and TH3 are configured as shown in Table 4.5, here IPC_{HtoL} is condition to shift to low-energy mode, IPC_{LtoH}

and $\#BR$ are condition to make high-speed mode. Because a higher value of IPC_{HtoL} means that the VSP processor is easier to unify pipeline stages and a lower value of IPC_{LtoH} indicates that it is hard to shift back to deeper pipeline, the VSP tends to run under low-energy mode for a longer time as it goes TH0 to TH3. The author experimentally decided these thresholds from around the average IPC under high-speed mode and these thresholds differ from thresholds used in Section 4.5.2.

74 SimPoints (84%) conform with the above theory. Figure 4.24 shows 16 representatives for the 74 major SimPoints. As shown in Fig. 4.24, the energy consumption is gradually reduced as thresholds are changed as TH0 to TH3. For these applications, the VSP can adjust the energy consumption according to processor workload or required response time.

On the other hand, all remaining 14 SimPoints show the tendency shown in Fig. 4.25 which shows all minor trend of SimPoints. The VSP executes these SimPoints under high-speed mode most of the execution time except for fixed low-energy mode because the VSP achieves higher IPC in the minor SimPoints than in the major SimPoints. In particular, as for gzip_2 and bzip_5486 the VSP reaches almost the highest IPC even in high-speed mode. For this reason, the VSP cannot gradually reduce the energy consumption for these SimPoints by using the combinations of the thresholds. However, since a longer pipeline is effective for the minor SimPoints, pipeline unification obtains less effectiveness compared with the major SimPoints. In terms of energy consumption, only 35% energy can be reduced at a maximum in the minor SimPoints while the VSP can reduce at least 40% energy in the major SimPoints. This means that the minor SimPoints have a small impact in energy reduction; thus, the behavior in the minor SimPoints is not a big matter for energy optimization.

Table 4.5: Thresholds configuration.

	IPC_{HtoL}	IPC_{LtoH}	$\#BR$
TH0	15	18	6
TH1	15	21	6
TH2	18	21	6
TH3	18	24	6

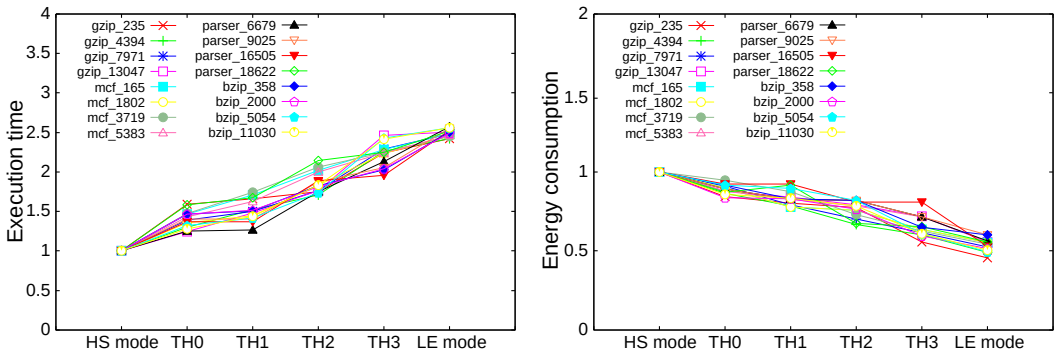

Figure 4.24: Major trend in 88 SimPoints.

For the major SimPoints, these thresholds are good to optimize the energy consumption. In contrast, for the minor SimPoints, although these thresholds cannot gradually reduce the energy consumption, the VSP cannot originally reduce a large energy consumption and have a small margin for energy optimization. Therefore, deciding thresholds from around the average IPC is appropriate for energy optimization. Using thresholds such as shown in Table 4.5, the VSP can optimize the energy consumption in major programs.

4.6 Glitch Propagation Problem

Besides depth-changing optimization, variable-depth pipeline architectures suffer from a common problem: glitch propagation increase. Variable depth

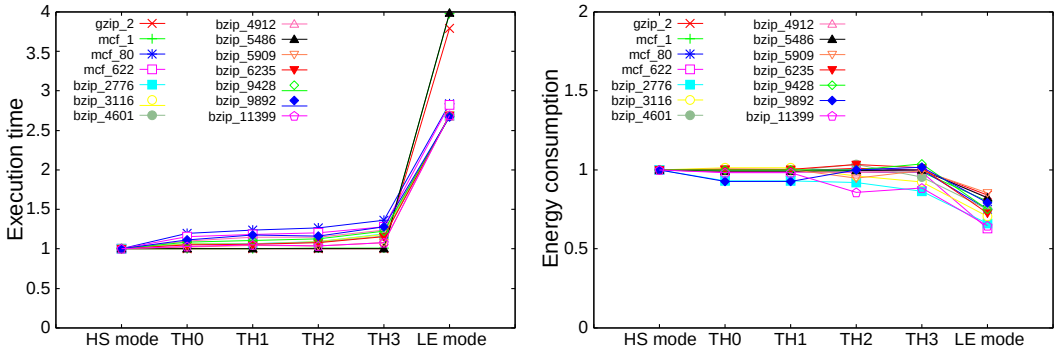

Figure 4.25: Minor trend in 88 SimPoints.

pipeline architectures use the D-flip-flop+multiplexer scheme shown in 4.17. Using the scheme, however, glitch propagation caused by stage unification becomes a serious problem, where a glitch is an unnecessary transition in performing a computation. While a functional transition, which is necessary for performing a computation, occurs either once or not at all (the signal remains unchanged) in each cycle, glitches can occur multiple times during a clock cycle. This means glitches consume a large amount of energy. Generally, glitches are generated by irregular delays of logic gates and wires, and the frequency of glitch propagation increases exponentially in proportion to the scale of the combinational circuits. Because multiple pipeline stages are unified and they have a large combinational circuit, the energy dissipation caused by glitch propagation increases exponentially.

4.7 Latch D-flip-flop selector-cell (LDS-cell)

To prevent glitch propagation, VSP introduces a special cell, the LDS-cell, as a pipeline register instead of D-flip-flop+multiplexer scheme. Figure 4.26 shows the circuit schematic of the LDS-cell, and Fig. 4.27 schematically illustrates its function. Under high-speed mode, the LDS-cell outputs a D-flip-

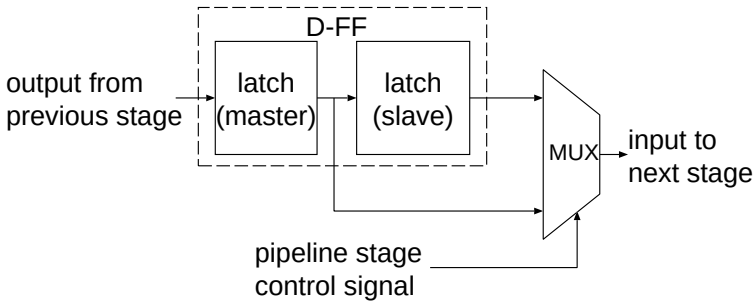

Figure 4.26: Latch D-flip-flop selector-cell (LDS-cell).

flop signal, consequently the LDS-cell behaves as a general pipeline register. In contrast, under low-energy mode, the LDS-cell outputs a master D-latch signal to prevent glitches. The functioning of the LDS-cell under low-energy mode is as follows.

- In the first half of the clock period, the master latch maintains data to prevent the glitches from propagating to the next stage.
- In the second half of the clock period, the master latch passes the input data to start the operations of the second half stages.

Therefore, the LDS-cell functions as a D-latch. Since the LDS-cell uses a master latch included in the D-flip-flop, the number of transistors in an LDS-cell is the same as a pair of a D-flip-flop and MUX.

4.7.1 Low-energy LDS-cell

The LDS-cell reduces the energy consumption to prevent glitch propagation, but the extra clock distribution for LDS-cells increases the energy consumption under LE mode. Compared to the PSU, the VSP can reduce the energy consumption when the effectiveness of the LDS-cell is larger than the overhead of the LDS-cell. However, the conventional LDS-cell has a large

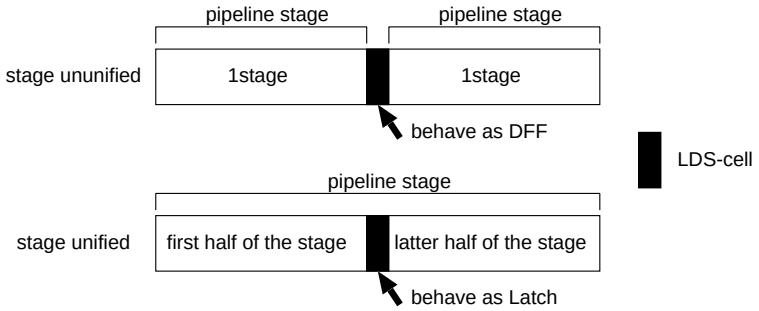

Figure 4.27: Function of LDS-cell.

overhead and the VSP cannot reduce the energy consumption more than the PSU under some conditions. Therefore, to reduce the overhead, the author propose two low-energy techniques for the LDS-cell that are introduced in the fabricated VSP chip.

Gated clock to LDS-cell

In a conventional VSP, a clock is supplied in the LDS-cell in each cycle, regardless of necessity, to simplify the processor design. Therefore, a gated clock is introduced into the LDS-cell. Typically, a gated clock is implemented to keep the clock at a low level. However, the author introduced a gated clock that keeps the clock at a high level. Because the LDS-cell prevents glitch propagation when the clock is at a high level, the clock is kept to such a level during clock-gated cycles, so the LDS-cell also prevents glitches when a gated clock is applied as shown in Fig. 4.28.

LDS-cell based on high-performance semi-static TSPC D-flip-flop

The conventional LDS-cell uses a static master-slave D-flip-flop, which is commonly used in standard cell based design. The static master-slave D-flip-flop contains a D-latch, so the LDS-cell can be implemented with the same

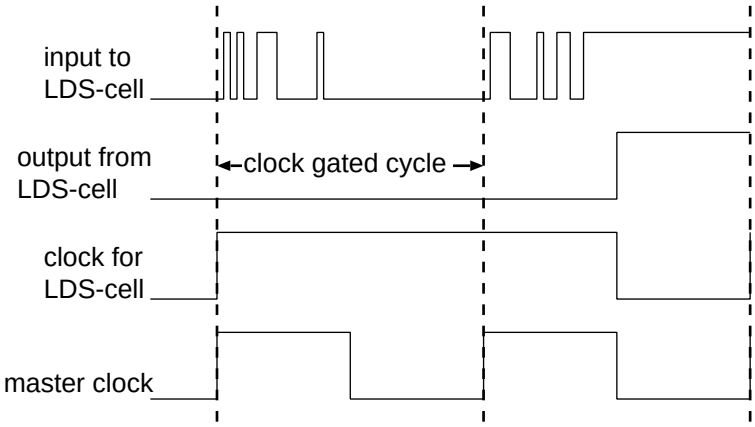

Figure 4.28: Timing diagram of gated clock for LDS-cell.

number of transistors as D-flip-flop+MUX. However, the static master-slave D-flip-flop has the drawback of large energy consumption.

Dynamic D-flip-flops, such as the Hybrid Latch Flip-Flop (HLFF) used by AMD K6 [51], DEC Alpha 21264's D-flip-flop [52] and Semi-Dynamic Flip-Flop (SDFF) [53] used by Sun UltraSPARC-3 are proposed, and they have the advantage in performance. However, the author does not use them as the base D-flip-flop for the LDS-cell because these D-flip-flops do not contain a D-latch; a D-flip-flop, D-latch and MUX are required to implement the LDS-cell. If an LDS-cell is designed based on these dynamic D-flip-flops, the hardware cost will increase.

In contrast, the high-performance semi-static true-single-phase clocking (TSPC) D-flip-flop (HSTSPC D-flip-flop) proposed in reference [54] contains a D-latch and achieves higher performance and lower energy consumption than the conventional D-flip-flops such as the static master-slave D-flip-flop, TSPC D-flip-flop and semi-static D-flip-flop.

The features of the HSTSPC D-flip-flop are as follows.

- A TSPC structure is adopted, which has the advantage of less clock energy consumption.
- The semi-static circuit can constantly keep data when the clock is at a low level, so it can solve the problem of using a dynamic circuit that cannot keep data with a gated clock.

Therefore, the HSTSPC D-flip-flop was introduced as a pipeline register including the LDS-cell. Figure 4.29 shows the circuit diagram of the LDS-cell based on HSTSPC D-flip-flop. From Fig. 4.29, we can understand that the LDS-cell can be implemented with the same number of transistors as the D-flip-flop+MUX.

The power of the LDS-cell based on HSTSPC D-flip-flop was evaluated using Rohm 0.18 μm CMOS technology and Synopsys HSPICE for a simulation. Figure 4.30 shows the results of the evaluation. In the figure, conventional static D-flip-flop indicates a static master-slave D-flip-flop, conventional LDS-cell indicates an LDS-cell based on the static master-slave D-flip-flop, and the HSTSPC LDS-cell indicates an LDS-cell based on the HSTSPC D-flip-flop. The vertical axis is power and the horizontal axis is the toggle rate of input data. Figure 4.30 shows that the HSTSPC LDS-cell can achieve lower energy than the conventional LDS-cell. Generally, pipeline registers have low toggle rates and the HSTSPC LDS-cell reduces energy consumption especially at a such toggle rate.

Problem and proposed solution

If both low-energy techniques are used simultaneously, then short-circuit current occurs in the LDS-cell and the energy consumption is significantly increased. The details of this problem and proposed solution are described as follows.

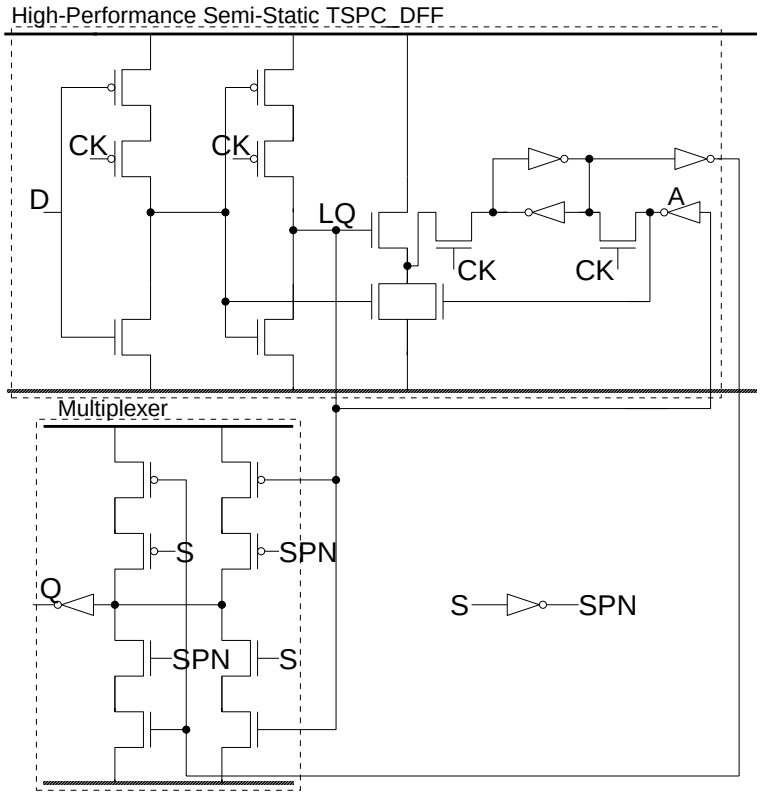

Figure 4.29: Circuit diagram of HSTSPC LDS-cell.

HSTSPC D-flip-flop has a master latch using dynamic circuit, which is constructed by connecting two clocked inverters, called a static p-stage. Node LQ in Fig. 4.29 is the output of the master latch and LDS-cell under LE mode. In this circuit, node LQ becomes a floating node when both the clock and input D are at a high level. Therefore, if the clock stays high when input D is high, a voltage drop occurs in node LQ because node LQ discharges with time. If the voltage of node LQ drops close to the threshold voltage, a short-circuit current occurs at inverter A in Fig. 4.29, thereby the energy consumption is significantly increased. Analysis of this voltage drop using

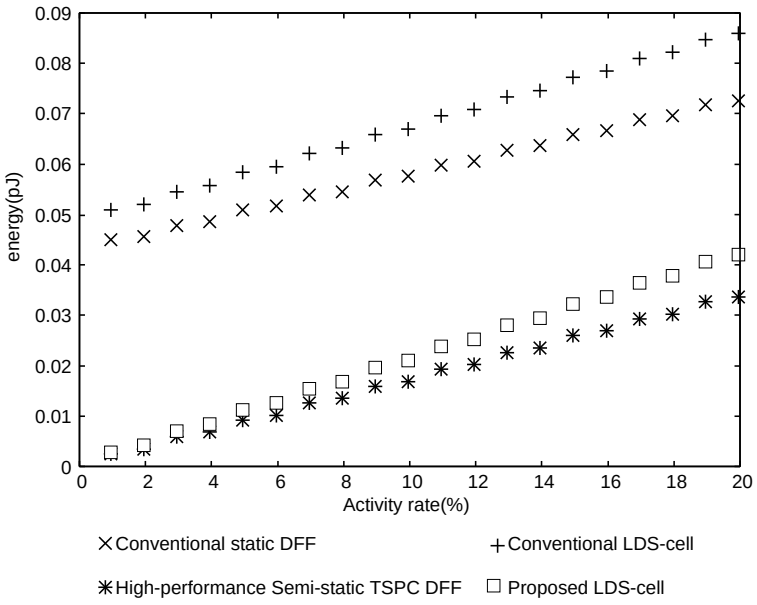

Figure 4.30: Power of HSTSPC LDS-cell.

Synopsys HSPICE shows that it took 400 μ seconds to begin, due to the flowing short-circuit current, which means there is no problem during typical operation. However, if both low-energy techniques are used simultaneously and the clock remains to at a high level for a long time, then the voltage drop becomes a serious problem.

Therefore, to solve this problem, the author proposes to add a PMOS keeper transistor as shown in Fig. 4.31. The PMOS keeper charges the node LQ, whether the clock is at a high or low level, so that no voltage drop occurs. In addition, this solution has only a small overhead in terms of the energy consumption, layout area and delay. The energy consumption increases by approximately 5% when compared with the HSTSPC LDS-cell. However, this increase is less than 0.1% of the overall energy consumption of the processor. In addition, the layout area is not increased because the number of

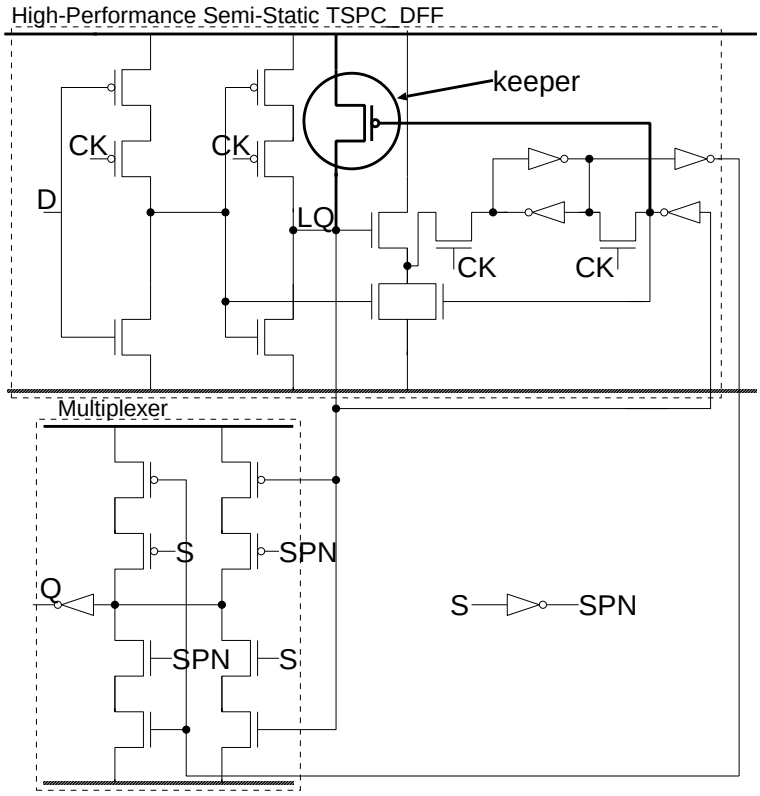

Figure 4.31: Circuit diagram of proposed LDS-cell.

PMOS transistors is less than that of NMOS transistors in HSTSPC D-flip-flop and there is dead space in the layout. Moreover, evaluation of the delay caused by addition of a PMOS keeper transistor shows that the delay does not increase.

For this reason, the author proposes a novel LDS-cell with a keeper (Fig. 4.31) as the solution to the voltage drop problem.

4.8 Evaluation Result

This section presents the effectiveness of proposed LDS-cell. Firstly, the author shows a simulation result using transistor-level simulation, then shows an actual measurement result using fabricated chip to demonstrate that LDS-cell saves larger energy on a chip than simulation.

4.8.1 Result of SPICE simulation

To evaluate the VSP processor, the author designed two processors; one uses the PSU technique and the other uses the VSP technique. The VSP uses low-energy techniques, as described in the previous section, and all pipeline registers of the PSU are used HSTSPC D-flip-flop similar to VSP.

The energy consumption was estimated using the Synopsys Nanosim tool. Both processors were designed to operate at 100 MHz for HS mode and 25 MHz (a quarter of 100 MHz) for LE mode. Figure 4.32 shows the energy consumption under LE modes normalized to the PSU result. The VSP also achieves approximately 5 to 7% less energy consumption than the PSU under LE mode. To break down the energy overhead and the energy reduction by using the LDS-cell, the author analyzes the energy of the LDS-cell, clock network and combinational circuit by using a fast SPICE tool. According to the evaluation results, due to the extra clock network and LDS-cells, the VSP has 3% energy overhead compared with the PSU. However, the VSP can reduce energy consumption by 8 to 10% compared with the PSU, so overall the VSP can reduce energy consumption by 5 to 7% more than the PSU. Under LE mode, the clock network consumes 15% of the energy of the processor compared with 30% under HS mode. This means the energy consumption of the combinational logic including glitches becomes more dominant. For this reason, preventing glitch propagation would be effective for energy reduction.

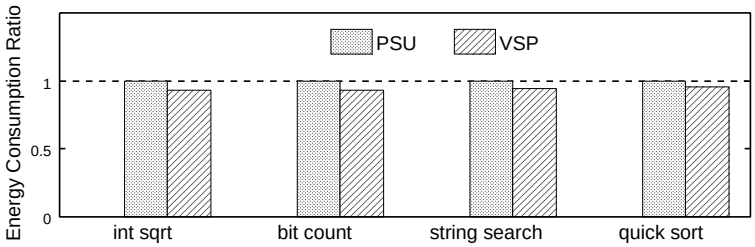

Figure 4.32: Result of SPICE simulation.

The wiring capacitance and wiring resistance were not accounted for in this simulation, due to the computational complexity. For this reason, this simulation can evaluate only glitches caused by gate delay. If the VSP is evaluated including wiring capacitance and wiring resistance, then the author surmises that VSP can prevent the propagation of glitches caused by wire delay, and that VSP should achieve higher efficiency than the present simulation results. Therefore, the author fabricated and measured both VSP and PSU chips.

4.8.2 Evaluation of fabricated chips

Figure 4.33 shows comparison of the energy consumption between the VSP and PSU under LE mode with each value normalized according to the PSU results. According to the results, the energy consumption of VSP is approximately 13% less than that of the PSU. Compared with the simulation results shown in Fig. 4.32, the effectiveness based on actual measurement with the fabricated VSP chip is larger. As mentioned above, the wiring capacitance and wiring resistance were not considered due to computational complexity, and only the glitches caused by gate delay were evaluated using the SPICE simulation. However, the measurement results show that VSP can also prevent glitches caused by wire delay. Therefore, the LDS-cell can reduce energy

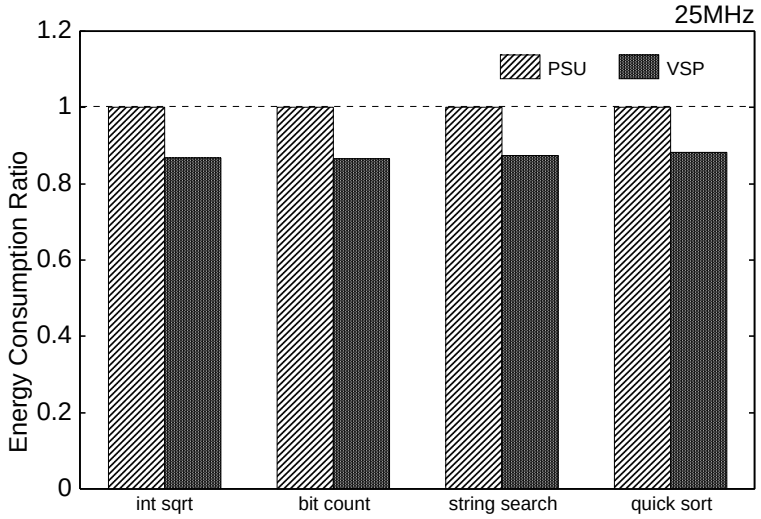

Figure 4.33: Measurement result of fabricated chip.

consumption further than that shown by the simulation results. The results for the fabricated chips show that VSP is superior to PSU, and that the effectiveness of the LDS-cell is higher than shown in simulation results from our former reports.

4.9 Summary of VSP processor

Developing energy-efficient processor core is a key for single-ISA heterogeneous multi-core processor design. The measurement results conduct that VSP architecture streamlines finer-grain diversity in a program phase using fine-grain depth-changing controller and reduces further energy with LDS-cell under folded pipeline structure. VSP is definitely suitable as a core architecture used in heterogeneous multi-core processors.

5 FabHetero

This chapter gives the detail of FabHetero project. Section 5.1 provides the motivation for automating heterogeneous multi-core design. Section 5.2 introduces an automated core design tool, called FabScalar, developed by research collaborator. Section 5.3 overviews FabHetero project. Sections 5.4 and 5.5 describe sub-projects in the FabHetero project. Section 5.6 summarizes FabHetero project.

5.1 Facilitating the Design of Heterogeneous Multi-core Processors

The studies of single-ISA heterogeneous multi-core processors attract much attention in various fields from mobile to high-performance computing because it streamlines the execution of diverse programs and program phases exploiting diversity of applications by providing diverse core types on a chip. Each core on a chip may differ in fetch/issue width, pipeline depths, size of tables to expose instruction-level parallelism (ILP), function unit mix, and structures of caches. Previous works in the heterogeneous multi-core architecture intend to obtain significant performance and power advantages. However, a practical issue currently impedes proliferating emerging single-ISA heterogeneous multi-core processors, namely, design and verification effort multiplied by the number of different design types of components in a chip. This factor limits the design space of microarchitectural diversity which can be practically implemented. The issue motivates us to automate single-ISA heterogeneous multi-core design in a meaningful design space to accelerate research and development productivities.

5.2 FabScalar

N. K. Choudhary et al. propose FabScalar to quickly design many cores that differ in the major superscalar design: superscalar width, pipeline depth, and sizes of structures for extracting ILP. FabScalar is the first solution for the design effort issue. FabScalar can immediately design diverse superscalar core types from a canonical pipeline stage library.

The canonical superscalar processor defined by FabScalar is shown in Fig. 5.34. FabScalar consists of nine canonical pipeline stages: Fetch, Decode, Rename, Dispatch, Issue, Reg. Read, Execute, Writeback, and Retire. FabScalar contains many synthesizable RTL designs for canonical pipeline stage, that differ in superscalar width, pipeline depth and size of tables. The front stages of FabScalar (Fetch to Dispatch) can be changed their width from 1-way to 8way. The Issue stage is sub-pipelined up to 3 stages and the Reg. read stage is sub-pipelined up to 4 stages respectively. In addition, size of stage-specific structures for extracting ILP such as issue queue, load and store queues, physical register file and reorder buffer are parameterized in the RTL. The current FabScalar tool-set quickly designs diverse superscalar core types within the above design space by editing parameter file. N. K. Choudhary et al. did validation experiments using 12 cores generated by FabScalar: functional and performance (instruction per cycle) validation and timing validation (cycle time) [11].

5.3 FabHetero Project Overview

FabHetero is the entire project of automatic heterogeneous multi-core processor generation. Although FabScalar contributes to mitigate the design and verification effort as regarding cores on chip, each core on a single-ISA heterogeneous multi-core requires a suitable cache system and each differently-

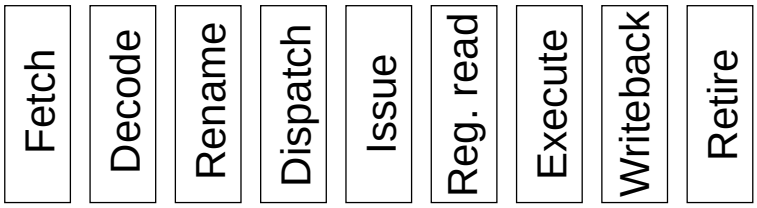

Figure 5.34: Canonical superscalar processor.

designed cache should be connected with a flexible shared bus system. As for the cache systems, designers must consider the organization including capacity, line size, associativity, hierarchy, and coherency protocol to optimize the design. In addition, the shared bus system should have a high flexibility because it connects diverse cache systems with a shared memory (the last level cache or main memory). These factors still block designing single-ISA heterogeneous multi-core processors in a short term even if core design is automated.

The author proposes a framework called FabHetero which automatically generate diverse heterogeneous multi-core processors using FabScalar, FabCache, and FabBus. Figure 5.35 shows an example of the heterogeneous multi-core processor generated by FabHetero framework. There are four differently-designed cores (Core 0, Core 1, Core 2 and Core 3), and each core has different cache structure. The shared bus connects the diverse cache systems with the shared memory (the last level cache or main memory). As for the cache systems, Core 1 has only L1 instruction and L1 data caches, Core 2 has dedicated L2 caches in addition to L1 caches, and Core 3 has unified L2 cache instead of dedicated L2 caches (each cache design may differ in cache capacity, line size, and associativity). Both L1 and L2 cache hierarchies can be omitted if a core does not require cache system as Core 0. Such non cache design is valuable for in particular the field of embedded system in which a

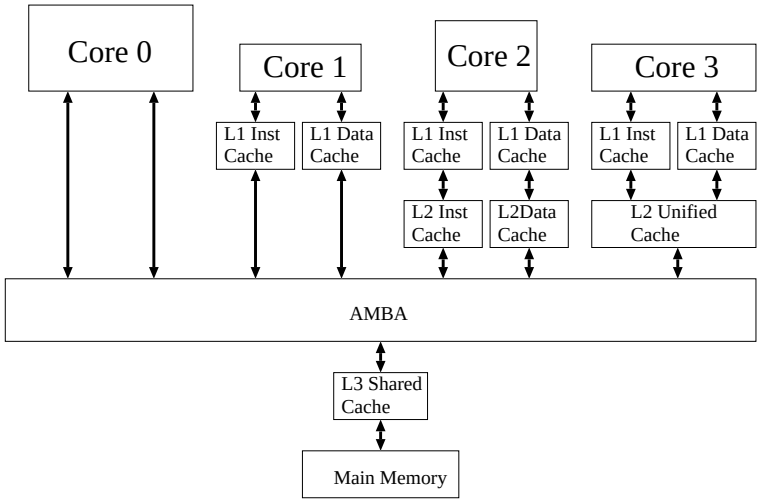

Figure 5.35: An Example of Heterogeneous Multi-core processor in FabHetero framework.

part of or all cache hierarchies are often removed. To generate various processor cores, cache systems and flexible shared bus system, FabHetero uses FabScalar, FabCache, and FabBus, respectively.

FabHetero confines the microarchitectural diversity into a superset code in SystemVerilog, this means that all parameterized microarchitectural diversity shares the same source file in RTL implementation. An opposite approach is using a generation script to generate RTL code, in this method, a generation script parses parameters and generates the target RTL code dedicated to the given parameters. Using generation script has the advantage of code optimization for each parameter compared with the superset strategy. However, using generation script has a critical problem when we (both developers and users) want to add a new microarchitectural approach. Since this work intends to use FabHetero for developing a new microarchitecture for heterogeneous multi-core processors, providing extensibility is essential

for our goal. Therefore, this work adopted superset strategy to implement FabHetero framework. By using superset strategy, adding a new microarchitecture becomes easier because we can directly implement it into RTL code while using generation script requires back-annotation to make the generation script after implementing it once. However, the superset strategy has inherent two concerns: the unintended hardware remains in the design and the code may diminish readability if the superset code naively includes various design choice. In FabCache implementation, for example, we implemented direct mapping as 1-way set associative to keep RTL code simple. As a result, extra circuits used to implement LRU remain in the design regardless of direct mapping. However, according to logical synthesis, the increase in area of a cache is only 0.76%, and we judged that the overhead is reasonable in order to maintain the readability. We have been implementing FabHetero avoiding the concerns. More detail of superset strategy is described in FabCache's paper.

A part of FabHetero project is now on fabrication stage, and the detail will appear in the future work.

5.4 FabCache

In this section, the author describes the microarchitectural diversity of FabCache.

5.4.1 FabCache Specification

Table 5.6 summarizes the available microarchitectural diversity in the current FabCache. The first column identifies the cache hierarchy. The second column shows ranges of line size, set size, and associativity. Set size can take on arbitrary value in all hierarchies. Line size has some limitations because

Table 5.6: Available designs in FabCache

Memory hierarchy	Dimensions(L = line size, S = set size, W = associativity)	Specific microarchitectures
L1 instruction cache	L = (fetch width to 2^n)×4(byte) S = 1 to 2^n W = 1, 2^n-way, full	**two banks interleaved** vs. non-interleaved **1 to 8 fetch width** interface with L2 cache **Line size transmission** vs. **burst transmission** **enable** vs. **disable**
L1 data cache	L = (1 to 2^n)×4(byte) S = 1 to 2^n W = 1, 2^n-way, full	Miss handling **blocking** vs. non-blocking writing approach **write-through** vs. write-back Interface with L2 cache **line size transmission** vs.**burst transmission** **enable** vs. **disable**
L2 cache	L = wider than higher hierarchy S = 1 to 2^n W = 1, 2^n-way, full	**dedicated instruction and data** vs. unified cache coherency **MOESI** vs. MOSI vs. MEI vs. **dedicated for each processor core.** interface with shared memory **processor num to/from one** vs. processor num to/from multi-ported (or multi-banked) memory cache replacement policy **LRU** vs. Pseudo-LRU **enable** vs. **disable**

of guaranty for completing cache operation from a higher hierarchy by one cache access. All cache hierarchies vary the associativity from direct mapping to 2^n-way set associative including full associative.

The third column in table 5.6 considers specific microarchitectural diversity for each hierarchy. Specific designs that are implemented in the current FabCache are highlighted in boldface in the third column. We also enumerate notable examples to emphasize the potential for FabCache in the future.

L1 instruction specific microarchitectures

Because FabScalar requires any instruction bundle (even instructions not aligned on cache line boundaries) every cycle, the instruction fetch requirement straddles two cache lines. As a result, getting all necessary instructions takes two cycles by twice access to SRAM in case of using single-ported SRAM because of the alignment limitation. This results in loss of instruction fetch efficiency and, consequently, the whole performance degrades. To

solve this problem, we implemented the L1 instruction cache as an interleaved memory. Interleaved memory is a technique to divide a memory into some banks to facilitate the simultaneous access to the memory banks. Therefore, we interleaved L1 instruction into two banks to adopt the instruction fetch requested from FabScalar. In addition, FabCache generates the interleaved caches for every fetch width (one to eight) of FabScalar.

L1 data specific microarchitectures

Currently, the miss handling strategy of L1 data cache is blocking whereas the processor core dispatches speculative load instructions to the L1 data cache. Non-blocking cache may improve the peak performance than the blocking cache, however, in terms of the energy consumption, preventing speculative loads from being dispatched to lower cache saves unnecessary energy consumption. Therefore, we consider it would be of interest to parameterize the diversity for targeted instruction level behavior. In the future, we will implement non-blocking and write-back mechanisms to cover a larger design space.

L2 cache specific microarchitectures

L2 cache can select two types: dedicated instruction/data or unified structure to design flexible cache systems. Caches in a multi-core processor should ensure cache coherency, and there are many cache coherency protocols such as MEI and MOSI. In addition, the suitable cache coherency protocol may be different for characteristic in a program. Firstly, we implemented MOESI protocol because the others can be implemented using MOESI structure. Choosing various coherency protocols helps us to explore the best coherency protocol.

Common microarchitectures

Interface with a higher level cache (or main memory) can choose up to line size from 1 word. Line size transmission sends data by line size unit at once. We can use the line size transmission for an on-chip communication which requires a wide bandwidth. Burst transmission sends data according to user defined bus width in each cycle until the all data in a line are transmitted. Although line size transmission requires a lot of I/O pins, the number of I/O pins is often not enough for the line size transmission. For this reason, we provide these transmissions as a parameter for chip fabrication to enable designers to choose the suitable type (and width) of transmission.

As cache replacement policy of N-way set associative cache, we adopt least recently used (LRU). We will implement other replacement algorithms to use a proper replacement algorithm for characteristic in a program. By enabling/disabling each cache, the cache hierarchy can be changed. When we enable a cache hierarchy, FabCache generates the cache according to parameter file. When we disable the cache, the cache directly accesses an ideal main memory using DPI-C and behaves as a perfectly hit cache for simulation. The disable mode is also favorable to estimate the best performance for each hierarchy.

5.5 FabBus

This section presents the microarchitectural diversity of FabBus.

5.5.1 FabBus Specification

Table 5.7 shows the configurable parameters in FabBus. The upper limit of number of master/slave unit depends on AMBA specification. In case that number of master unit is only one, any units for cache coherency control are

Table 5.7: FabBus Specification

Number of Units	
Master Unit	1∼16
Slave Unit	1∼16
Snoop Bus	0∼16
Cache Coherency Control	
Way to Assert Signals	Broadcast One-to-One Wired AND/OR
MOESI	confirmed
MESI	confirmed
Berkeley, etc.	unconfirmed
Bus Width	
Data Width	256 [bit]
Address Width	64 [bit]
Configurable Function	
Arbitration Algorithm	Fixed Priority [or] Round Robin

not generated. Arbitration algorithm can be chosen from fixed priority and round robin. Proposed bus framework enables to design various shared buses which work on multi-core processor.

5.6 Summary of FabHetero project

The author framed FabHetero framework to automatically generate diverse heterogeneous multi-core processors. Currently, FabHetero automatically generates diverse processors including a superscalar core, its cache system up to level-two, and bus system. Our laboratory has started to use FabHetero as a development environment in our other researches, and we will release FabHetero for developers and researchers in the near future to boost

research for heterogeneous multi-core processor.

6 Co-simulation Framework

This Chapter presents a practical processor development framework used in FabHetero. Section 6.1 provides overview of this work. Sections 6.2 and 6.3 introduce the current environments to research processor architecture. Section 6.4 overviews proposed co-simulation framework using functional and timing simulators. Section 6.6 details external system call emulator. Section 6.7 presents checkpoint mechanism in detail.

6.1 Requirements for Rapid Processor Prototyping

As multi-core architecture has become commonly used to improve processor performance, designing a state-of-the-art multi-core chip in a short time has become essential for processor research. A development environment that contains useful mechanisms and can be used throughout the entire processor research provides efficient infrastructure to researchers. The steps of fabricating a novel processor chip are classified into five phases, 1) design space exploration using a simulator, 2) register transfer level (RTL), 3) gate level, 4) transistor level, and 5) fabricated chip. There are two challenges to streamline the processor development through the entire standard ASIC design flows.

System Call Emulation in RTL through fabricated chip

When researchers prototype a processor from RTL, to gate and transistor level, to ASIC, it is often desirable to focus on user level code because they are interested in the core part of the processor and not all of the system level support. They may be in this situation because they designed the RTL from scratch or because they are using open source toolsets (e.g., FabScalar) which provide a level of sophistication in the microarchitecture but do not

currently feature system level support. As a matter of convenience, and a matter of research productivity, it is good to dispense with the issue of explicitly supporting system calls in the processor design. While emulation is often used in simulators written in a high-level language, it is unwieldy to carry that over to RTL/gate/transistor simulations and not at all possible to emulate in the same way for a fabricated chip.

Turnaround Time Reduction for Evaluation and Verification

Except for the fabricated chip phase, all other simulation-based phases in particular gate/transistor level cannot simulate the entire workloads in a reasonable timeframe. Therefore, a checkpoint mechanism is needed to resume a simulation from an arbitrary region of interest (ROI). Moreover, checkpoints are useful when hardware bugs are detected in the fabricated chip. A checkpoint allows for bypassing the bug (if it is infrequent) to get to another ROI. Therefore, it is also useful for validation in the fabricated chip phase.

6.2 Processor Simulators

Many processor simulators [55, 56] and system simulators [57–59] written in a high-level language are used for processor research. Researchers take advantage of such simulators in the early stage of research in accordance with their intended use. Since main focus is from RTL to fabrication, in which researchers evaluate the precise hardware cost, energy efficiency, and circuit delay for their proposed approach, three focused mechanisms are described: 1) system call emulation to simplify processor architecture, 2) checkpoint mechanisms to reduce simulation time, and 3) cache warming mechanism to achieve a highly accurate evaluation. These mechanisms, however, are used only in each simulator. The goal is to use these mechanisms in all phases of

standard ASIC design flows.

6.3 Synthesizable Processors

Some open synthesizable processors can be used from RTL implementation to chip fabrication [11, 59–61]. Since FabScalar and OpenSPARC have a co-simulation environment, the author describes these two processors in more detail.

FabScalar automatically generates synthesizable RTL designs of differently designed superscalar cores. FabScalar contains an instruction set simulator, called functional simulator, to verify RTL by concurrently running the same instructions in RTL design and the functional simulator, and cross-checking the architectural state instruction-by-instruction. The functional simulator is also used for emulating a system call, so RTL design can handle a system call as a one-cycle-instruction. Also FabScalar provides fast-skip and checkpoint mechanisms to avoid long simulation time and re-simulating up to a checkpoint. However, FabScalar currently has drawbacks in system call emulation (described in Section 6.6) and the checkpoint mechanism (described in Section 6.7). The aim was to improve the two mechanisms based on FabScalar.

OpenSPARC is the open-source version of UltraSPARC T1 and T2 processors. Currently, RTL design, simulation tools, and verification package are all available. OpenSPARC provides a complete RTL design to boot a full OS and useful tools for simulation and verification including checkpoint and cache warming. However, there is the level of abstraction gap as the next step from a processor simulator, and this gap makes it difficult to advance the research phase beyond simulator-based exploration. By contrast, proposed off-chip system call emulation mechanism enables a designer to evaluate a

prototype processor omitting touchy hardware for an OS with general benchmark programs. In addition, cache warming of OpenSPARC is implemented by programming language interface (PLI) in verilog, this limits the use to only in RTL simulation. Proposed cache warming mechanism is unique in that it is consistently used from RTL to fabricated chip.

6.4 Co-simulation Framework

This subsection gives an overview of co-simulation framework. The framework consists of a *functional simulator* written in a high-level language and *processor design*. Note that *processor design* refers to all designs of RTL, gate, transistor level, and fabricated chip. Fig. 6.36 shows how the functional simulator is used in the co-simulation framework. The functional simulator assists in the verification and evaluation of processor design. The cross-checking architectural state guarantees instruction set level behavior of processor design, and fast-skip and checkpoint mechanisms reduce turnaround time. In addition, the functional simulator emulates system calls by calling the host OS according to a request from processor design.

6.5 Challenges

Three challenges in the co-simulation framework are described below.

System call emulation: System call emulation is significantly beneficial in that a designer can run a general program without booting an OS on the target processor. Proposed system call emulation mechanism can be used for every research phase. The co-simulation framework exploits general load and store instructions to communicate with the emulator; therefore, no special mechanism is necessary in the processor design.

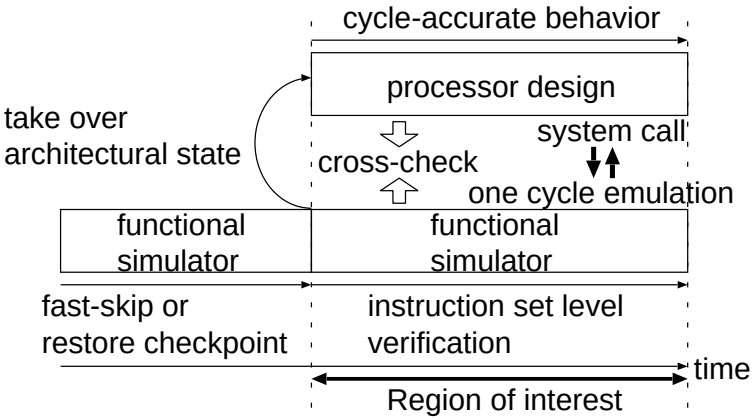

Figure 6.36: Co-simulation framework.

Checkpoint mechanism: The checkpoint mechanism is used not only to reduce turnaround time but also to evaluate only an ROI. Checkpoint creation for a multiprocessor should consider the non-deterministic problem. The author proposes a checkpoint mechanism that solves this problem.

Cache warming: Proposed checkpoint mechanism contains cache warming mechanism as a part of optional checkpoints. Restoring a checkpoint is exposed to a large performance gap with a peak performance because the simulation is resumed with a cold started cache. In addition, in the gate and transistor level phases, it takes a long time to achieve the peak. Proposed cache warming mechanism warms up the cache in the shortest time and improves evaluation accuracy.

The framework is introduced into two processor design projects: an embedded processor [22] and FabScalar.

6.6 External System Call Emulator

6.6.1 System call emulation

In general benchmark programs such as SPEC, a processor must handle system calls (services from an OS kernel) to handle the file system, network, memory, process, thread, and security. For this reason, to evaluate a processor design with general benchmarks, the processor either boots an OS or uses an alternative stand-alone C library. There are two requirements. One is that researchers directly execute benchmarks on a full implemented processor to evaluate and verify in a short time. Since booting an OS takes a large amount of time, especially in gate and transistor level, it is difficult to evaluate or verify a processor design on a full system. Moreover, although using an LSI tester has an advantage of directly evaluating or testing a fabricated chip with input vectors, such LSI testers limit the execution cycle up to insufficient cycles for running a target application. The other requirement is that to evaluate a microarchitectural approach, researchers often require only primal instructions such as arithmetic, logical, branch, and memory access instructions, and tends to omit subsidiary hardware for an OS such as memory management unit and internal processor registers. Implementing such hardware requires researchers to have a deep understanding of such hardware.

Because of these two requirements, executing general programs without an OS is valuable. Newlib is a C library intended for use on embedded systems [62]. A processor can execute programs without an OS with the addition of a few low-level routines. However, another binary file using Newlib is needed. In addition, Newlib requires emulation of peripheral systems and does not support multiple processes and cores.

Emulating a system call as an instruction solves the above problems.

When a system call occurs, the functional simulator detects the call and emulates it by calling the host OS. Later, the processor design continues execution after reflecting the result of the system call.

To emulate a system call, the processor design somehow notifies the emulator of the occurrence of the system call. Furthermore, the processor design must take over the system call result. In the RTL phase, this is not difficult because a test module can look into the submodule, which asserts a relative signal, and overwrite the architectural state. FabScalar currently uses this method; however, it is used only in the RTL phase and prevents regions including system calls from being evaluated on FPGA [63]. Therefore, the co-simulation framework is necessary for emulating system calls beyond the RTL phase.

6.6.2 Implementation of off-chip system call emulator

The concept of system call emulation mechanism is juggling a system call as consecutive stores and loads. Proposed emulation mechanism is explained using Figs. 6.37 and 6.38. Fig. 6.37 shows memory mapping and how to trigger/reflect a system call emulation. A memory space (e.g., from address *7fd00000*) is allocated to interact with the off-chip emulator. First, when a system call occurs, the processor jumps to the system call trap routine like a real product. The routine shown in Fig. 6.38 is used instead of a true routine if an user wants to emulate system calls. Second, the processor design involves storing the architectural state, i.e., register file, to the prescribed space because the emulator requires the register file values to emulate the required system call. Next, the emulator emulates the system call when a store is executed into the probing address (*7fd00000*). Finally, the emulator writes the values updated by the system call into the same memory space to

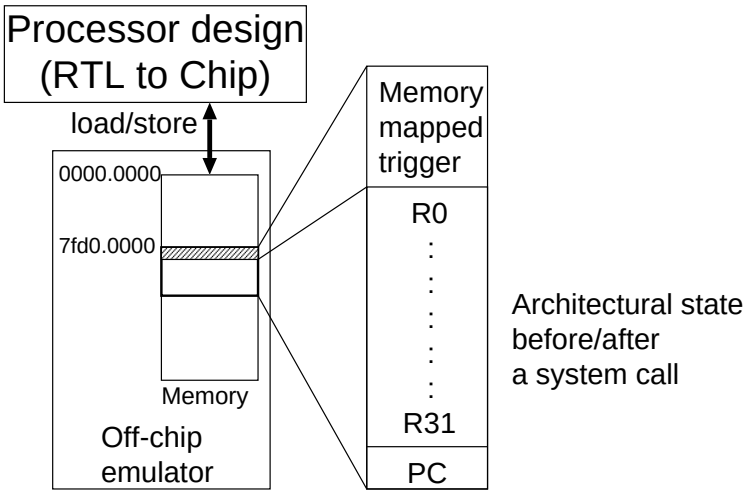

Figure 6.37: Off-chip system call emulation mechanism.

which the processor stored the register values, then the processor loads the modified values using load instructions. Note that if the processor includes a cache, the stores and loads should be non-cacheable memory access instructions. This emulation mechanism does not require any dedicated hardware in the processor design; therefore, the processor maintains a pure design. Since the processor interacts with the emulator using load and store instructions in the co-simulation framework, the emulation mechanism can be consistently used from RTL to fabrication. This enables the processor design to execute a general program without booting an OS.

The off-chip system call emulation mechanism is of course used for evaluating and verifying a complete processor. Also, a prototype processor design which does not support system calls can be evaluated with general benchmarks. This aspect improves research productivity to evaluate a microarchitectural approach.

```
bfc003d0 <__trap_syscall>:
        /* Save architectural state */
        sw $1,  0x104(k0)
                  :
        sw $31, 0x17c(k0)
        /* Trigger for system call */
        sw 0x01, 0x0000(k0)
        /* Restore the result */
        lw $1,  0x104(k0)
                  :
        lw $31, 0x17c(k0)
```

Figure 6.38: System call trigger routine

6.7 Checkpoint Mechanism Solving Non-deterministic Problem

6.7.1 Essential State Restoration

A checkpoint mechanism saves the state of a simulation in an ROI and later continues the simulation from the ROI. A checkpoint mechanism goes through two phases: checkpoint creation phase with only the functional simulator, and resume phase with the processor design. We can repeat the resume phase during verification and evaluation of a processor to reduce turnaround time.

In proposed co-simulation framework, we can resume a program using a similar routine as the system call emulation for use in every design phase to restore the architectural state. The reset routine shown in Fig. 6.39 is used. After the processor is reset, the program counter is initialized to *bfc00000*, which is the general start address of the reset routine. In the routine, the processor loads register file values and the program counter written into the prescribed memory space. The program counter indicates the starting point of a checkpoint.

However, if the co-simulator naively resumes a benchmark program from

```
bfc00000 <__reset_handler>:
        lui k0,  0x7fd0
        lw  $1,  0x104(k0)
                 :
        lw  $31, 0x17c(k0)
        /* load program counter */
        lw  k1,  0x278(k0)
        jr  k1
```

Figure 6.39: Reset routine

a checkpoint, a system call (file- and network-related) cannot be correctly executed. In the following explanation, a sequence of file system operations is used as an example to simplify the problem. The co-simulator leaves file input/output (I/O) to the OS running on a host computer. Fig. 6.40 shows the issue of resuming simulation from a checkpoint. When a file is opened, the off-chip system call emulation mechanism calls the OS to handle the file opening (Fig. 6.40.A). Once the file is opened, the co-simulator treats I/O operation to the file in the same way (Fig. 6.40.B). Once the simulation reaches at the start point of an ROI, the co-simulator creates the checkpoint, then the file is closed because the co-simulator quits (Fig. 6.40.C). For this reason, when the co-simulator resumes the simulation from the checkpoint (Fig. 6.40.D) and a file I/O occurs (Fig. 6.40.E), the co-simulator cannot handle the file I/O because the file is not open.

To solve this problem, FabScalar dumps the state not only at a checkpoint but also at file I/Os in an ROI as shown in Fig. 6.41. In the checkpoint creation phase, FabScalar executes a program beyond a checkpoint to dump the state after file I/Os in the ROI (Fig. 6.41.A). To resume a program, FabScalar restores the state at the checkpoint (Fig. 6.41.B). When a file I/O occurs during the resumed simulation, the dumped state after the file I/O is restored (Fig. 6.41.C); therefore, FabScalar reproduces the state after

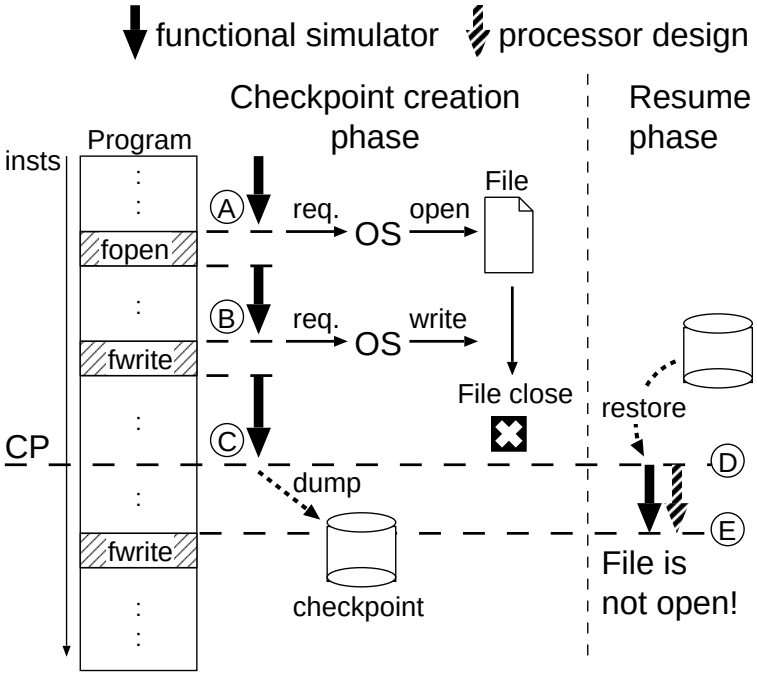

Figure 6.40: Problem with checkpoint mechanism.

the file I/O. With this method, FabScalar provides a checkpoint mechanism. However, FabScalar can execute only file I/Os that were pre-executed in the checkpoint creation phase. In addition, this mechanism cannot be applied to a multiprocessor environment because we cannot create the preceding checkpoint. Because the execution order is non-deterministic in a multiprocessor, the order of system calls is also non-deterministic.

There are a few solutions to the problems. M5 uses the solution of dumping all necessary information into a checkpoint file, e.g., the offset of the file descriptor manipulated in the simulation. By contrast, we adopt another solution because M5's solution requires the dumping all inflight operations such as file system and network. In our solution, the simulator executes only related system calls to resume a program with inflight operations.

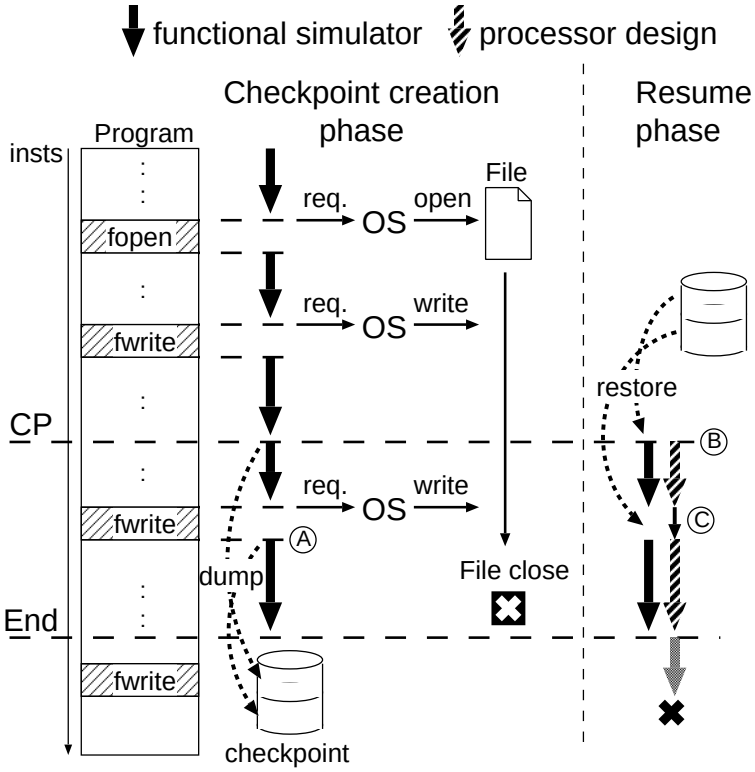

Figure 6.41: Checkpoint mechanism of FabScalar.

Fig. 6.42 shows the solution. The co-simulator dumps the difference in state between each file I/O up to a checkpoint in the checkpoint creation phase (Fig. 6.42.A). It skips the instructions between each file I/O using the dump file and executes only file I/Os (Fig. 6.42.B). After it reaches the checkpoint, it continues execution including file operations (Fig. 6.42.C). As a result, the co-simulator skips to a checkpoint at high speed without any restriction.

Although the solution has the advantage of handling all inflight operations in the same way, restoration speed depends on the number of system calls up to a checkpoint. To demonstrate that the solution is practical, the author evaluated the restoring of speed using SPEC2000 INT benchmarks. The

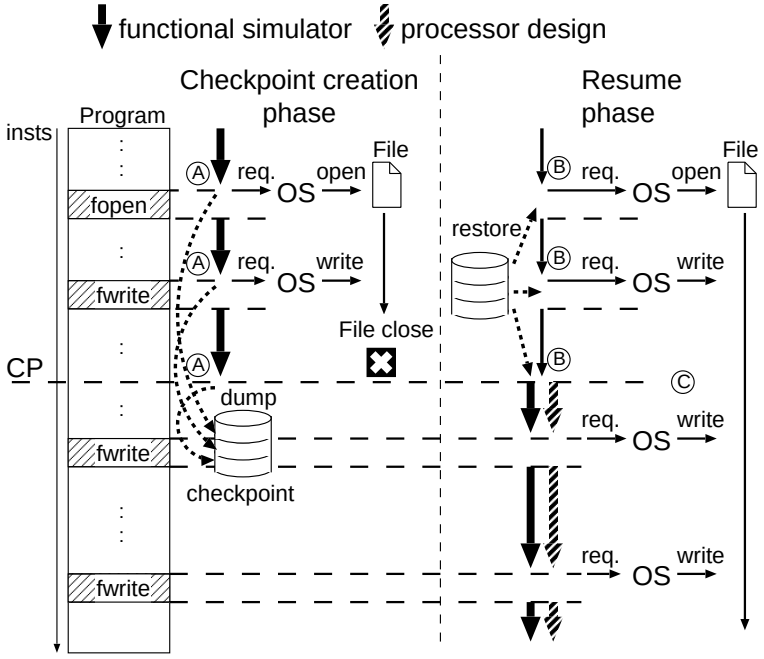

Figure 6.42: Proposed checkpoint mechanism.

author created a checkpoint in SimPoint [64] and resumed the checkpoint for each benchmark program. Table 6.8 lists the evaluation results. The upper half compares the time to forward each benchmark to the SimPoint. The author evaluated all benchmarks on Intel Core i7-2600 CPU @ 3.40 GHz with 4 GB memory. The author used a five-stage single pipeline processor design as the RTL design. The lower half of the table summarizes the number of skipped instructions, the file size of the checkpoint, and the number of system calls up to the SimPoint. The results show that restoring a checkpoint took a few seconds in the worst case and the file size of the checkpoint was not so large.

Table 6.8: Demonstration of checkpoint mechanism.

	gzip	mcf	bzip	parser	twolf
RTL design[a] (hour)	2026	784	1718	1632	1550
fast-skip (min.)	244	103	206	210	212
checkpoint (sec.)	0.50	0.68	0.77	3.84	0.53
skipped insts (100 million)	1189	553	977	1146	1066
checkpoint file size (MB)	832	326	384	1780	119
system calls	65	116	101	1027	133

[a]Estimated by million instructions per second (MIPS) value

6.7.2 Optional State Restoration

A cache system has a large impact on processor performance. When we resume a benchmark program from a checkpoint, a cold started cache incurs a performance gap with peak performance, as shown in Fig. 6.43. Therefore, the evaluation accuracy is degraded because of the performance gap. Furthermore, it takes a long simulation time to warm up a cache system to analyze the peak performance/energy. OpenSPARC has a cache warming mechanism using PLI in verilog HDL. This implementation limits the use to only in the RTL phase. By contrast, proposed cache warming mechanism can be used in all design phases. It is particularly effective in shortening the test vector for an LSI tester. In addition, the cache warming mechanism defines a certain time when the cache system is warmed up, this feature enables a designer to evaluate only a specified period after the processor achieves the peak in simulation, as shown in Fig. 6.44.

Fig. 6.44 shows the cache warming mechanism. The co-simulator also has a cache simulator written in C language. When the co-simulator creates a checkpoint, the cache system dumps the cache warming routine (binary

file, actually), as shown in Fig. 6.44. Lines that are accessed with the same index are dumped in order of the least recently used (LRU) value to restore the cache contents including the cache replacement algorithm. A lower LRU value has a higher priority for replacement, i.e., an entry whose LRU value is *0* will be replaced. The dumped routine is linked when the co-simulator starts restoration, and it is called in the reset routine before restoring the architectural state. This mechanism restores the cache contents by using a software level approach in the shortest time.

The author briefly estimated the impact of the cache warming mechanism on an L1 data cache (total size: 16 KB and line size: 16 bytes). Even in the worst case (cache warming on blocking cache), the cache warming mechanism reduced 90% of the simulation time to stabilize performance compared with a simulation using cold started cache. Proposed cache warming mechanism reduces one more order of magnitude when we use a non-blocking cache for cache warming.

Currently, optional checkpoints are only for data cache warming. Expanding optional checkpoints such as instruction cache and branch predictors is left for future work.

6.8 Summary of Co-simulation Framework

e proposed a co-simulation framework that provides system call emulation, checkpoint, and cache warming mechanisms through the RTL, gate level, transistor level, and fabricated chip phases. All the mechanisms were effective in two processor design projects.

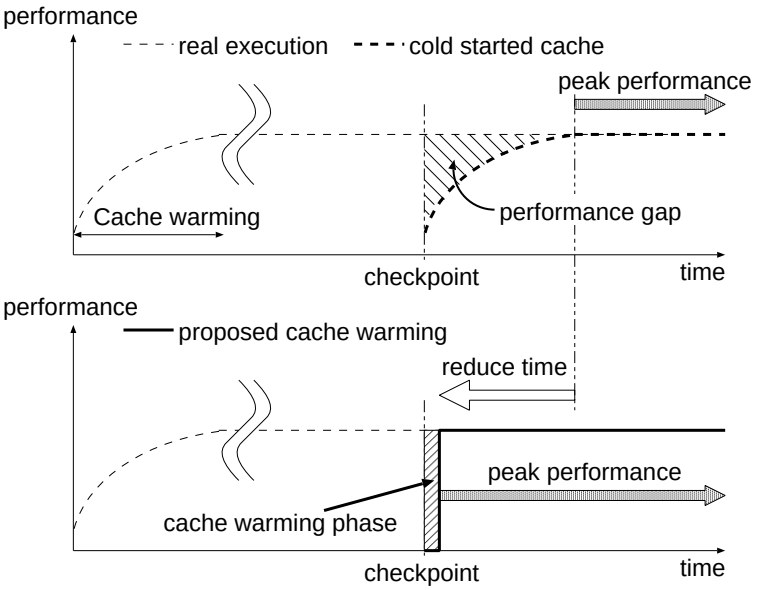

Figure 6.43: Impact on performance using cache warming.

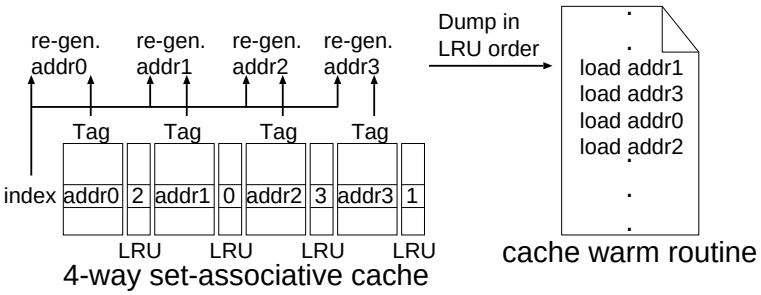

Figure 6.44: Generating cache warming routine.

7 Conclusion

7.1 Summary

This thesis addressed four big challenges to fabricate low-energy heterogeneous multi-core processors, development of low-energy D-flip-flops, enhancement of variable stages pipeline (VSP) architecture, framing FabHetero project, and establishing practical processor development environment, all works head to yield emerging single-ISA heterogeneous multi-core processors.

Since D-flip-flops dominate a large portion of delay, area, and energy in a processor, improving D-flip-flops yields a big benefit to enhance processor performance. The author proposed two differently-targeted D-flip-flops for a heterogeneous multi-core processor: one is for higher-performance core and the other is for lower-energy core. Both D-flip-flops achieve a higher-energy efficiency compared with the conventional D-flip-flops. The D-flip-flop emphasizing performance contributes to construct higher-performance cores, and the energy-efficient D-flip-flop facilitates producing lower-energy cores on a heterogeneous multi-core processor.

Even on heterogeneous multi-core with ideal scheduler, optimization of energy in a fine-grained term (hundreds or thousands of clock cycles) is unwieldy due to the overhead caused by application migration. In contrast, VSP architecture with proposed fine-grained depth-changing controller well streamlines fine-grain heterogeneity in a program phase. In addition, LDS-cell, a key component of VSP, further reduces the energy consumption under folded pipeline structure by preventing glitch from propagating. Fabricating VSP processor chip demonstrates that VSP architecture can be implemented by reasonable hardware cost, and LDS-cell has larger energy saving than sim-

ulation result.

A key problem of developing or researching heterogeneous multi-core processor is its large design effort; processor architects must design diverse cores, caches and bus system and try out a good combination. FabHetero helps to mitigate the design effort problem to design diverse cache systems and flexible shared bus system which compose a heterogeneous multi-core processor. The author employs an user-friendly implementation, *superset*, that will help proliferate emerging single-ISA heterogeneous multi-core processors. FabHetero provides the synthesizable register transfer level (RTL) designs of heterogeneous multi-core processors within in a design space (combinations of diverse cores, caches and bus system). This thesis provided the design space currently implemented. Diverse caches and shared bus system with superscalar cores generated by FabScalar are easily accessible to processor architects.

In an academic situation, fabrication, verification and evaluation of a processor chip are heavy tasks especially for a small research group. The author proposed a practical processor development environment which uses both functional simulator written in a high level language (C++) and timing simulator written in RTL (SystemVerilog). Researchers can enjoy the convenience previously provided only by software simulators on standard ASIC design flows. This feature improves research productivity, and chip fabrication is feasible by less effort for small research group.

Totally, the former two are for improvement of the energy efficiency of heterogeneous multi-core processor, and the latter two provide a development environment for proliferation of single-ISA heterogeneous multi-core processors. All works contribute to fabrication of low-energy heterogeneous multi-core processors for overcoming the current energy issue, leading more energy efficient computer systems.

7.2 Future works

As for research of VSP architecture, two advanced works will be required: (1) evaluating the effectiveness on diverse and complex microarchitectures, and (2) energy optimization as a core embedded in a heterogeneous multi-core processor. In this work, the author demonstrates the effectiveness of VSP architecture on a scalar pipelined processor, i.e., 7-stage pipeline on a MIPS R3000 compatible instruction set. Currently, superscalar architecture has become widely used even in mobile computers which are strictly constrained by energy consumption. For this reason, demonstrating the effectiveness of VSP on superscalar architecture is essential for VSP research. The work will require to try out different combinations of based superscalar implementation (fetch width, issue width, pipeline depth, and etc.) and VSP implementation (how many stages are unified, and unification control method). FabHetero will facilitate the work since it generates diverse superscalar processors. The other work will investigate how to control fluctuation of processor workload on a heterogeneous multi-core processor composed of VSP cores. Some interesting inquiries are as follows. Whether all cores adopt VSP technique or only some of them adopt that? Can we use conventional scheduling methods to obtain sufficient energy reduction or should we innovate a new scheduler which corresponds to the behavior of VSP? Providing answers for these questions will help producing more energy-efficient heterogeneous multi-core processors.

For design automation of heterogeneous multi-core processor, this work frames FabHetero framework and implements key parts, cache system and shared bus, in RTL language (SystemVerilog). Since designing a physical design from RTL is not easy task especially for complex superscalar architecture (e.g., how to implement highly multi-ported memory), providing certain

steps for chip fabrication is of value. Therefore, a processor generated by FabHetero will be fabricated, and it will demonstrate that chip fabrication of diverse heterogeneous multi-core processors is feasible by reasonable effort. In addition, increasing microarchitectural diversity in FabHetero is also important to provide more choices for processor architects. FabHetero, including co-simulation framework, will be released for researchers and developers in the near future to fuel more innovation of heterogeneous multi-core processor. The development environment makes diverse single-ISA heterogeneous multi-core processors more accessible to researchers and developers and improves research productivity of processor architecture on standard ASIC flows. After constructing the development environment for single-ISA heterogeneous multi-core processor, the research will advance to application field: considering how to design low-energy heterogeneous multi-core processor for target system. In the current framework, researchers should manually explore design space in FabHetero. Even though FabHetero facilitates getting processor design of diverse heterogeneous multi-core processors, the exploration may impose much effort on researchers, without practical guidance. As a future work, it is a worthwhile challenge that FabHetero automatically generates a processor design corresponding to desired performance. Currently, the parameter file of FabHetero requires designers to directly specify microarchitecture (e.g., fetch-width, issue-width, cache capacity, and etc.), consequently the designers must explore the design space in FabHetero with parameter tuning to find the most suitable parameters. Therefore, providing processor design having desired performance will be of value for processor designers.

Acknowledgments

This thesis consists of research products during the M.S. and Ph.D. programs in Graduate School of Engineering, Mie University, there has been constant support from countless people.

I would like to thank Dr. Takahiro Sasaki. I have learned many things from him throughout the M.S. and Ph.D. programs: computer architecture, presenting, managing research group, and English. In addition, thanks to him for providing valuable opportunities to fabricate VLSI chip. I would like to thank Dr. Toshio Kondo, his constructive advice has improved my researches, in particular developing low-energy D-flip-flops.

I has been a privilege to work with Dr. Eric Rotenberg, a wonderful researcher, of North Carolina State University in FabHetero project. Meaningful discussion with him has made FabHetero project better. Moreover, thanks to him for providing oversea internship in North Carolina State University.

I would like to thank Drs. Toshio Kondo, Yoshikatsu Ohta, and Kazuo Mori for serving on my advisory committee and providing incisive comments and constructive feedback to improve this dissertation.

This work is supported by the VLSI Design and Education Center (VDEC) of the University of Tokyo in collaboration with Synopsys, Inc., Cadence Design Systems, Inc., and Mentor Graphics, Inc. The VLSI chip in this study was fabricated in the chip fabrication program of VDEC in collaboration with Rohm Corporation and Toppan Printing Corporation. This work is supported by a Grant-in-Aid for Young Scientists (19700042) from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

Special thanks to English teachers in Graduate School of Engineering, Mie University for supporting my English learning. Thanks to all labora-

tory member, they provide opportunities of in-deep discussion related with research, sport, and refreshing to me.

Finally, I would like to thank my parents, Hiroyuki and Michiyo, instilling in me the value of education and giving me their unambiguous support throughout my life.

References

[1] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, K. I. Farkas. Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance. *31st International Symposium on Computer Architecture (ISCA31)*, pp. 64-75, June 2004.

[2] H. H. Najaf-abadi and E. Rotenberg. Configurational Workload Characterization. *International Symposium on Performance Analysis of Systems and Software 2008 (ISPASS-2008)*, pp. 147-156, April 2008.

[3] P. Greenhalgh. Big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7. ARM WHITE PAPER: http://www.arm.com/ja/files/downloads/big.LITTLE Final.pdf.

[4] A. Chandrakasan, W. J. Bowhill, and F. Fox. Design of high-performance microprocessor circuits, *IEEE Press*, Piscataway, NJ, 2001, 1st edn.

[5] R. Kusaba and T. Kondo. High-Performance Semi-Static TSPC DFF *IEICE J. of Electronics Society*, May, 1998, Vol. J-81-C-2, No. 5, pp. 469-476, (Japanese).

[6] D. Liu and C. Svensson. Power consumption estimation in CMOS VLSI chips, *IEEE J. Solid-State Circuits*, 1994, 29, (6), pp. 663-670.

[7] H. Kawaguchi and T. Sakurai. A reduced clock swing flip-flop (RCSFF) for 63% power reduction, *IEEE J. Solid-State Circuits*, 1998, 33, (5), pp. 807-811.

[8] I. Hong, M. Potkonjak, and M. A. Horowitz, On-line scheduling of hard real-time tasks on variable voltage processor, *Proceeding of International Conference on Computer-Aided Design*, pp. 653-656, November. 1998.

[9] J. Pouwelse, K. Langendoen, and H. Sips, Dynamic voltage scaling on a low-power microprocessor, *Proceeding of 7th ACM International Conference on Mobile Computing and Networking (Mobicom)*, July, 2001, pp. 251-259.

[10] I. Atsuki, Design Constraint of Fine Grain Supply Voltage Control LSI, *Proceeding of the 16 th Asia and South Pacific Design Automation Conference (ASP-DAC 2011)*, pp. 760-765, January, 2011.

[11] N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. H. Dwiel, S. Navada, H. H. Najaf-abadi, and E. Rotenberg, Fab-Scalar: Composing synthesizable RTL designs of arbitrary cores within a canonical superscalar template, *Proceedings of the 38th IEEE/ACM International Symposium on Computer Architecture (ISCA-38)*, pp. 11-22, June 2011.

[12] T. Nakabayashi, T. Sasaki, K. Ohno, and T. Kondo. Low Power Semi-static TSPC D-FFs Using Split-output Latch. *Proceedings of the International SoC Design Conference (ISOCC2011)*, pp. 167-170, November 2011.

[13] T. Nakabayashi, T. Sasaki, K. Ohno, and T. Kondo. VLSI implementation of Variable Stages Pipeline Processor using Fine-Grain Pipeline

Depth Controller. *Proceedings of the 27th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC2012)*, pp. C-T1-03, July 2012.

[14] T. Nakabayashi, T. Sasaki, H. Nakamura, K. Ohno, and T. Kondo. Measurement of Low-Energy Processor Chip using Fine-Grain Variable Stages Pipeline Architecture. *Proceedings of the 4th International Workshop on Advances in Networking and Computing (WANC2012)*, pp. 293-297, December 2012.

[15] T. Nakabayashi, T. Sasaki, H. Nakamura, K. Ohno, and T. Kondo. Energy Optimization using Fine-Grain Variable Stages Pipeline Processor Chip. *International Journal of Networking and Computing (IJNC)* Volume 3, No. 2, pp. 192-204, 2013.

[16] T. Sasaki, T. Nakabayashi, K. Nomura, K. Ohno, and T. Kondo. Design and Evaluation of Fine-grain Mode Transition Method based on Dynamic Memory Access Analyzing for Variable Stages Pipeline Processor. *IET Journal of Computers and Digital Techniques* (accepted).

[17] T. Nakabayashi, T. Sasaki, T., and T. Kondo. Dynamic BTB Resizing for Variable Stages Superscalar Architecture. *Proceedings of the First International Symposium on Computing and Networking Across Practical Development and Theoretical Research (CANDAR 2013)*, pp. ??-??, December 2013.

[18] T. Nakabayashi, T. Sasaki, K. Ohno, and T. Kondo Low Energy Techniques for Variable Stages Pipeline Focused on Clock System Power, *IEICE Journal of Information and System Society*, Vol. J94-D, No. 4, pp. 646-656, 2011. (Japanese)

[19] T. Nakabayashi, T. Sasaki, K. Ohno, and T. Kondo. Design and Evaluation of Variable Stages Pipeline Processor with Low Energy Techniques. *IET Journal of Computers and Digital Techniques*, Volume 6, Issue 1, pp. 43-49, 2012.

[20] T. Nakabayashi, T. Sasaki, E. Rotenberg, K. Ohno and T. Kondo. Research for Transporting Alpha ISA and Adopting Multi-processor to FabScalar. *Symposium on Advanced Computing Systems and Infrastructures 2012 (SACSIS2012)*, pp. 374-381, May 2012. (in Japanese)

[21] T. Nakabayashi, T. Sugiyama, T. Sasaki, E. Rotenberg, and T. Kondo. Co-simulation framework for streamlining microprocessor development on standard ASIC design flow. *Proceedings of the 19th Asia and South Pacific Design Automation Conference (ASP-DAC2013)*, pp. ??-??, January 2014.

[22] T. Sugiyama, T. Nakabayashi, T. Sasaki, and T. Kondo. Development of C++/RTL co-simulation environment for accelerating VLSI design of an embedded processor.g *Proceedings of the 28th International Technical Conferenc3 on Circuits/Systems, Computers and Communications (ITC-CSCC2013)*, pp. 281-284, July 2013.

[23] T. Okamoto, T. Nakabayashi, T. Sasaki, T., and T. Kondo. FabCache: Cache Design Automation for Heterogeneous Multi-core Processors. *Proceedings of the First International Symposium on Computing and Networking Across Practical Development and Theoretical Research (CANDAR 2013)*, pp. ??-??, December 2013.

[24] Y. Seto, T. Nakabayashi, T. Sasaki, and T. Kondo. FabBus: A Bus Framework for Heterogeneous Multi-core processor. *28th International*

Technical Conferenc3 on Circuits/Systems, Computers and Communications (ITC-CSCC2013), pp. 254-257, July 2013.

[25] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[26] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9, October 1974.

[27] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, Dark Silicon and the End of Multicore Scaling, *Proceedings of the 38th IEEE/ACM International Symposium on Computer Architecture (ISCA-38)*, pp. 365-376, June 2011.

[28] David Lammer. Intel cancels Tejas, moves to dual-core designs. EETimes article.

[29] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction. In*Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture, MICRO 36*, 2003.

[30] M. Aater Suleman, Onur Mutlu, Moinuddin K. Qureshi, and Yale N. Patt. Accelerating Critical Section Execution with Asymmetric Multi-core Architectures. *In Proceedings of the 14th international conference on Architectural support for programming languages and operating systems, ASPLOS 2009*, pp. 253-264, 2009.

[31] Tegra Whitepaper, Variable SMP-A Multi-Core CPU Architecture for Low Power and High Performance,

http://www.nvidia.com/content/PDF/tegra_white_papers/tegra-whitepaper-0911b.pdf.

[32] H. Partovi, R. Burd, U. Salim, F. Weber, L. DiGregorio and D. Draper. Flow-Through Latch and Edge-Triggered Flip-Flop Hybrid Elements, *Proceedings of IEEE International Solid-State Circuits Conference Digest of Technical Papers*, February 1996, pp. 138-139.

[33] F. Klass. Semi-dynamic and dynamic flip-flops with embedded Logic, *Proceedings of IEEE Symposium on VLSI Circuits Digest of Technical Papers 1998*, pp. 108-109, Jun 1998.

[34] R. Kumar, K. C. Bollapalli, R. Garg, T. Soni, and S. P. Khatri. A Robust Pulsed Flip-Flop and its use in Enhanced Scan Design, *IEEE Proceedings of International Conference on Computer Design 2009*, pp. 97-102, October 2009.

[35] H. Yingbo and Z. Runde LOW CLOCK-SWING TSPC FLIP-FLOPS FOR LOW-POWER APPLICATIONS, *World Scientific J. of Circuits, Systems and Computers*, Vol. 18, No. 1, pp. 121-131, 2009.

[36] J. Yuan and C. Svensson. High-speed CMOS circuit technique, *IEEE J. of Solid-State Circuits*, vol. 24, pp. 62-71, Febrary 1989.

[37] "PTM Homepage." http://ptm.asu.edu/.

[38] H. Sunagawa and H. Onodera. Variation-tolerant Design of D-FlipFlops, *Proceedings of IEEE International SOC Conference 2010*, PP. 147-151, September 2010.

[39] Y. Ichikawa, T. Sasaki, T. Hironaka, T. Kitamura, and T. Kondo, Low Energy Consumption by a Variable Stages Pipeline Technique, *Proceed-*

ing of International Technical Conference on Circuits/Systems Computers and Communications, July, 2004, 0358, 6C1L-4.

[40] T. Sasaki, Y. Ichikawa, T. Hironaka, T. Kitamura, and T. Kondo, Evaluation of Low Energy and High Performance Processor using Variable Stages Pipeline Technique, *IET Journal of Computer and Digital Techniques*, April, 2008, Vol. 2, No. 3, pp. 230-238.

[41] T. Sasaki, K. Nomura, T. Nakabayashi, K. Ohno, and T. Kondo, Fine Grain Controller for Variable Stages Pipeline Processor *International Technical Conference on Circuits/Systems, Computers and Communications*, July, 2010, pp. 748-751.

[42] H. Shimada, H. Ando, and T. Shimada, Pipeline Stage Unification: A Low-Energy Consumption Technique for Future Mobile Processors, *Proceeding of the International Symposium on Low Power Electronics and Design 2003*, August, 2003, pp. 326-329.

[43] H. Shimada, H. Ando, and T. Shimada, A Hybrid Power Reduction Scheme Using Pipeline Stage Unification and Dynamic Voltage Scaling, *Proceeding of the 9th IEEE Symposium on Low-Power and High-Speed Chips*, April 2006, pp. 201-214.

[44] J. Yao, S. Miwa, H. Shimada, and S. Tomita, A Dynamic Control Mechanism for Pipeline Stage Unification by Identifying Program Phases, *IEICE TRANSACTIONS on Information and Systems*, Vol. E91-D, No. 4, April 2008, pp. 1010-1022.

[45] J. Yao, S. Miwa, H. Shimada, and S. Tomita, A Fine-Grain Runtime Power/Performance Optimization Method for Processors with Adaptive

Pipeline Depth, *Journal of Computer Science and Technology*, Vol. 26, No. 2, pp. 292-301, Mar. 2011. DOI 10.1007/s11390-011-1132-9.

[46] J. Koppanalil, P. Ramrakhyani, S. Desai, A. Vaidyanathan, and E. Rotenberg, A Case for Dynamic Pipeline Scaling, *Proceeding of International Conference on Compilers, Architecture, and Synthesis for Embedded Systems 2002*, October, 2002, pp. 1-8.

[47] H. Onodera, A. Hirata, A. Kitamura, K. Kobayashi, and K. Tamaru, P2Lib:Process Portable Library and Its Generation System, *Journal of Information Processing*, vol.40, no. 4, pp. 1660-1669, April, 1999, (In Japanese).

[48] MiBench. http://www.eecs.umich.edu/mibench/

[49] Intel Corporation: Intel Atom Processor Z5xx Series Datasheet(2010)

[50] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically Characterizing Large Scale Program Behavior, *10th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2002, pp. 45-57.

[51] H. Partovi, R. Burd, U. Salim, F. Weber, L. DiGregorio, and D. Draper. Flow-Through Latch and Edge-Triggered Flip-Flop Hybrid Elements, *Proceedings of IEEE International Solid-State Circuits Conference Digest of Technical Papers*, February 1996, pp. 138-139.

[52] B. Nikolic, V. Stojanovic, V. G. Oklobdzija, W. Jia, J. Chiu, and M. Leung. Sense Amplifier-Based Flip-Flop, *Proceedings of IEEE International Solid-State Circuits Conference Digest of Technical Papers*, February 1999, pp. 282-283.

[53] N. Nedovic and V. G. Oklobdzija. Dynamic Flip-Flop with Improved Power, *Proceedings of International Conference on Computer Design*, September 2000, pp.323-326.

[54] R. Kusaba and T. Kondo. High-Performance Semi-Static TSPC DFF, *IEICE Journal of Electronics Society*, May, 1998, Vol. J-81-C-2, No. 5, pp. 469-476, (Japanese).

[55] D. Burger and T. M. Austin. The Simplescalar tool set, version 2.0. technical report, *CS-TR-1997-1342, University of Wisconsin-Madison*, 1997.

[56] R. Shioya, M. Goshima, and S. Sasaki. The design and implementation of processor simulator Onikiri2, *the Annual Symposium on Advanced Computing Systems and Infrastructures, poster*, 2009.

[57] F. bellard. QEMU: open source processor emulator. http://bellard.org/qemu/.

[58] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. The M5 simulator: modeling networked systems, IEEE Micro, 26:52-60, 2006.

[59] N. Fujieda, T. Miyoshi and K. Kise. SimMips: A MIPS system simulator, *Workshop on Computer Architecture Education held in conjunction with MICRO-42*, pp. 32-39, December 2009.

[60] XUM Version 2.0: the eXtensible Utah Multicore Project, September, 2012. http://www.cs.utah.edu/formal_verification/XUM/.

[61] OpenSPARC:
http://www.oracle.com/technetwork/systems/opensparc/index.html.

[62] Newlib
http://sourceware.org/newlib/.

[63] B. H. Dwiel, N. K. Choudhary, and E. Rotenberg. FPGA Modeling of Diverse Superscalar Processors, *Proceedings of the 2012 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'12)*, pp. 188-199, April 2012.

[64] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, Automatically Characterizing Large Scale Program Behavior, *10th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2002, pp. 45-57.