A Study of Time-multiplexed Execution Models for Dynamically Reconfigurable Processors

Yohei Hasegawa

A dissertation submitted in partial fulfillment of
the requirements for the degree of

DOCTOR OF PHILOSOPHY

School of Science for Open and Environmental Systems
Graduate School of Science and Technology
Keio University

2007

Preface

Field-programmable devices such as Field-Programmable Gate Arrays (FPGAs) allow us to implement any customized logic functions after chip fabrication and reuse the expensive silicon circuitry for different purposes. However, the FPGAs are generally perceived as area- and power-inefficient because they use a large number of transistors to provide high programmability. To cope with this problem, more cost- and power-aware dynamically reconfigurable processors have received attention as a flexible off-loading engine for versatile System-on-Chips (SoCs).

The dynamically reconfigurable processors have a two-dimensional array of 4-bit to 32-bit simple coarse-grained processing elements (PEs). Each PE has an internal memory to hold multiple circuit configuration data called a context, and PE operations and inter-PE connections can be dynamically reconfigured by rapidly switching contexts every clock cycle. In recent years, various architectures have been released, but a methodology or guideline to design an efficient architecture has not been established because of the wide design space compared to the FPGAs.

This dissertation describes hierarchical time-multiplexed execution models for dynamically reconfigurable processors and their evaluation results of performance, area cost, and power consumption. The goal of this study is to quantitatively evaluate the fundamental trade-offs and the area overhead associated with the time-multiplexed execution models and finally to provide a guideline for SoC designers to design area- and power-efficient architectures for a target application.

At first, the context-level time-multiplexed execution is characterized, and we propose an estimation method of performance and cost for various PE-array sizes. Evaluation result based on NEC Electronics' Dynamically Reconfigurable Processor (DRP) shows that the model can estimate the most area- and power-efficient PE-array size though there exist differences between the model and actual measurement. Next, the task-level time-multiplexed execution model, in which computational tasks can be switched on-demand, is investigated based on practical applications.

Finally, MuCCRA-1, which supports the hierarchical time-multiplexed execution models, is implemented in Rohm 0.18μm CMOS technology, and its performance and area overhead are evaluated. On a 5.18mm-squared die, 4×4 24-bit PE array, four multipliers, and four distributed memory modules are implemented. Each PE has a 64-depth context memory, and the context and task switches are controlled by efficient control mechanisms. Evaluation result demonstrates that the area overhead required for the control mechanisms of time-multiplexed execution models is only about 1.3% of the total area. In contrast, the area of a context memory in each PE amounts to 55.7% of the PE area.

Acknowledgments

I would like to thank all the people who supported to accomplish this thesis.

First and foremost, I would like to express my deepest gratitude to my supervisor, Professor Hideharu Amano for providing me this precious study opportunity as a Ph.D student in his laboratory. He kindly approved my request of participation to the laboratory although I was a graduate in economics. He has always given many beneficial advices and suggestions to my research. His great experience guided me to enlarge my knowledge on the computer architecture fields.

I would like to thank all of my doctoral committee members, Professor Yoshikazu Yamamoto, Professor Tadahiro Kuroda, and Associate Professor Shingo Takada for their careful reviews and comments to my thesis. I also would like to thank Associate Professor Nobuyuki Yamasaki for his technical comments.

I am very grateful to many advisers in NEC/NEC Electronics Corporation. I especially would like to thank Dr. Kenichiro Anjo (Texas Instruments Japan Ltd.) and Dr. Toru Awashima (NEC EDA R&D Center) for their technical supports and valuable comments to my research. I could not achieve this work without their kindness supports. I also would like to thank Koichiro Furuta, Toshiro Kitaoka (NEC Electronics), Takao Toi (NEC EDA R&D Center), Nobuki Kajiwara, Hirokazu Kami, and Takeshi Inuo (NEC IP Platform Laboratory) for their technical supports of DRP design tools.

I also would like to thank JST research project leaders, Professor Kimiyoshi Usami (Shibaura Institute of Technology), Associate Professor Hiroshi Nakamura (University of Tokyo), and Associate Professor Mitaro Namiki (Tokyo University of Agriculture and Technology), for their valuable comments to the MuCCRA-1 implementation.

I will forever be thankful to my former research adviser, Professor Mikio Nakayama, for providing advice many times during my graduate school career.

I especially would like to appreciate my senior, Naoto Kaneko (Nokia Research Center, Japan), for his support to my internship in Nokia Research Center and fruitful advices and suggestions to my research. I also would like to thank my seniors, Associate Professor Yuichiro Shibata (Nagasaki University), Assistant Professor Michihiro Koibuchi (National Institute of Informatics), Dr. Konosuke Watanabe, Dr. Yasuki Tanabe (Toshiba Semiconductor Company), Dr. Yasunori Osana, and Tomohiro Otsuka for many insightful technical discussions and comments.

I am grateful to my friend, Yutaka Yamada (Toshiba R&D Center), for his technical advices,

fruitful discussions, and kind encouragement. He is also one of my best friends in off-campus activities. Athletics and karaoke with him were great stress releases for my rushed campus life.

This work could not be achieved without many collaborators. I would like to express my appreciation to my colleagues who worked with me in the DRP project: Katsuaki Deguchi, Shohei Abe (Nomura Research Institute), Masayasu Suzuki, and Syunsuke Kurotaki (Sony Corporation).

I especially would like to express my sincere thank to Satoshi Tsutusmi for his valuable comments and suggestions to my research. His efforts provide significant improvement and advancement for the MuCCRA project. The MuCCRA-1 chip could not be fabricated without his technical supports, suggestions, and especially MuCCRA-editor. I also would like to thank Vasutan Tunbungheng, Vu Manh Tuan, Takuro Nakamura, Takashi Nishimura, Adepu Parimala, Toru Sano, Masaru Kato, Shotaro Saito, Keiichiro Hirai, Naomi Seki, and all of the research collaborators of WASMII Group.

I would like to extend many thanks to my colleagues: especially Akira Tsuji, Hiroki Matsutani, and Masato Yoshimi. I have been receiving daily inspiration and encouragement from the profitable discussions with them. I am grateful to Yuri Nishikawa for her careful English support. I wish to thank all the members of Amano Laboratory for their daily supports during my campus life. I also wish to thank all the members of Anzai-Imai Laboratory and Yamasaki Laboratory.

My special thank goes to Kumi Nanjo (Office of Research Administration, Keio University) for her careful fund management and support.

Finally, I am grateful to my parents and family for giving me constant support and encouragement throughout my life and always having faith in me.

This research was supported by the funding of NEC Corporation and Japan Science and Technology Agency (JST). I would like to thank Japan Society for the Promotion of Science (JSPS) for the research fellowship (DC2). The DRP-1 device and design tools were provided by NEC Electronics Corporation and NEC Corporation.

I would like to thank VLSI Design and Education Center (VDEC), the University of Tokyo for providing chip fabrication services and CAD tool supports. I especially wish thank Associate Professor Makoto Ikeda (VDEC) for his management in VDEC and many technical supports. The MuCCRA-1 chip in this study has been fabricated in the chip fabrication program of VDEC in collaboration with Rohm Corporation and Toppan Printing Corporation. The CAD tools which are used in this work have been supported by VDEC in collaboration with Synopsys, Inc., Cadence Design Systems, Inc., and Mentor Graphics, Inc.

Yohei Hasegawa

Yokohama, Japan
August 2007

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The increasingly higher integration of transistors at an increasingly lower cost per transistor has resulted in a capability of putting over a billion transistors on a single chip. This progress has led designers to use a System-on-a-Chip (SoC) design methodology in a wide variety of application areas. Recently, SoCs have been in widespread use as a total solution for single-chip system integration. The components in such SoCs may include embedded microprocessors, memory blocks, external interfaces, peripherals, and application-specific customized functional blocks.

However, recent embedded products such as cell phones, game machines, and digital appliances require integrating more and more functionality on a chip with lower power, smaller area, and lower overall cost. The performance of such devices depends on application-specific customized blocks, but they are not so flexible to be reused and standardized for various products. Hence, SoC designers need to design many kinds of SoCs for each target application. In order to achieve shorter turnaround time, a system-level design methodology based on reusable Intellectual Property (IP) cores and high-level synthesis technologies has emerged. With deep sub-0.1μm CMOS technology, however, back-end design phases including place and route, chip layout, and physical verification are dominantly time-consuming procedures. Therefore, even with the current SoC design methodology, it is significantly difficult to design SoCs in high-mix low-volume production.

Field-programmable logic devices are alternative flexible hardware to Application Specific Integrated Circuits (ASICs). Field-Programmable Gate Arrays (FPGAs) [1,2] allow us to implement any customized logic functions after chip fabrication and reuse the expensive silicon circuitry for different purposes. However, the FPGAs are generally perceived as area- and power-inefficient because they use a large number of transistors to provide high programmability, and several approaches to cope with this problem have been proposed [3,4].

As dynamically reconfigurable devices have a more cost- and power-aware architecture, they have received attention as a flexible off-loading engine for versatile SoCs [5, 6]. Most current dynamically reconfigurable processors [7, 8] consist of a two-dimensional array of 4-bit to 32-bit

coarse-grained processing elements (PEs) connected by a switch-based network. The PE used in dynamically reconfigurable processors is composed of a simple shift and mask unit, an arithmetic and logic unit, and a register file. Each PE has a memory component to hold multiple circuit configuration data called context, and each context is constructed by operational instructions for the PE and connection instructions for routing resources. By rapidly switching contexts, the PE operations and intra/inter-PE connections can be dynamically reconfigured every one or a few clock cycles.

In general dynamically reconfigurable processors, a target computational task can be divided into multiple contexts. Each reconfigurable element such as PE has a context memory for storing many contexts. The dynamically reconfigurable processors can read a context in its context memory according to a context pointer delivered from a context controller. This multi-context dynamic reconfigurability provides higher area- and power-efficiency compared to traditional FPGAs due to the time-multiplexed PE array. In this thesis, we refer to this execution scheme based on the multi-context dynamically reconfigurable processor as time-multiplexed execution.

In the time-multiplexed execution, the PE-array size defined by the number of PEs per context is a primary architectural factor which strongly influences the speed, area, and power consumption of the dynamically reconfigurable processor. The PE-array size determines the degree of time-multiplexing, and there exist trade-offs between performance and area/power associated with the PE-array size. In the last decade, different architectures have been released by various companies so far, but a methodology or guideline to design an efficient architecture, especially the PE-array size, for its target purpose has not been established because of its wide design space compared to FPGAs.

1.2 Objective and Contribution

This thesis focuses on studying the hierarchical time-multiplexed execution models for multi-context dynamically reconfigurable processors. Through trade-off evaluations of the time-multiplexed execution models, we investigate and suggest a guideline to design an area- and power-efficient architecture for its target application.

At first, a context-level time-multiplexed execution model [9–11] is investigated. This work proposes a parallelism-based analytical method to estimate the optimal PE-array size, and performance and area/power trade-offs are quantitatively evaluated based on NEC Electronics Dynamically Reconfigurable Processor (DRP).

Secondly, a task-level time-multiplexed execution model is discussed. For supporting huge applications over context memory capacity and multiple applications, this model enables multiple tasks to be executed in the time-multiplexed manner without disturbing the PE-array. Since how to instruct the whole device for a target application is important in this model, several DRP-based experiments considering practical applications [12–16] are carried out as case studies.

Final work of this thesis describes MuCCRA-1 architecture [17, 18] which supports both the context- and task-level time-multiplexed execution models completely. The MuCCRA-1 has been

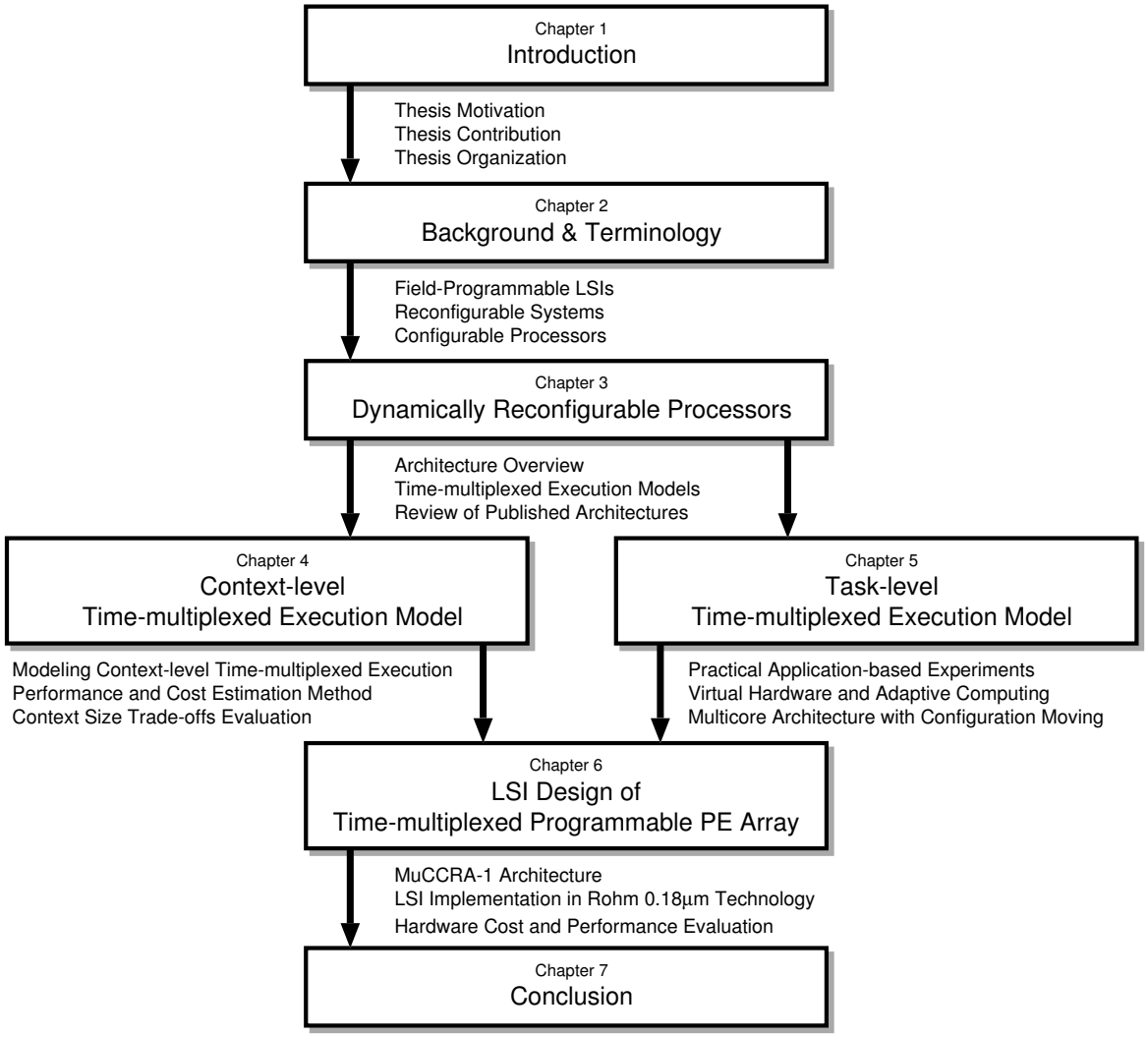


Fig. 1.1: Thesis Organization

fabricated in Rohm 0.18μm CMOS technology. A 4×4 24-bit PE array, four dedicated multipliers, and four distributed memory modules are integrated onto a 5.18mm-squared die. Each PE has a 64-depth context memory, and context and task managements are controlled by simple controllers. This work demonstrates quantitative evaluation results of area overhead and performance of the time-multiplexed execution models.

1.3 Thesis Organization

This thesis is organized as follows, and the thesis organization is illustrated in Fig. 1.1.

Chapter 2 reviews a technical background of field-programmable technologies and reconfigurable systems. After introducing a traditional FPGA architecture model and current Xilinx Spartan-3A FPGA, several reconfigurable and configurable architectures which include basic technologies of dynamically reconfigurable processors are briefly described.

Chapter 3 describes an overview of dynamically reconfigurable processors and time-multiplexed execution models focused throughout this thesis. The chapter also includes brief introductions of currently released device architectures. The DRP architecture assumed in Chapter 4 and Chapter 5 is also described in this chapter.

Chapter 4 provides a context-level time-multiplexed execution model. This chapter includes a parallelism-based analytical method to estimate the optimal PE-array size in area- and power-efficiency. The performance and area/power trade-offs are quantitatively evaluated based on the DRP architecture.

Chapter 5 discusses a task-level time-multiplexed execution model. This chapter reports three practical DRP-based experiments including a cryptographic accelerator for IP security (IPsec) with a virtual hardware mechanism, an adaptive Viterbi decoder, and a data-resident multi-core structure.

Chapter 6 describes a physical implementation of MuCCRA-1 architecture. In this chapter, after introducing the MuCCRA-1 architecture and control mechanisms, performance and cost evaluations for the time-multiplexed execution models on the MuCCRA-1 are shown.

Chapter 7 concludes this thesis and suggests a potential avenue for future exploration.

Chapter 2

Background and Terminology

This chapter describes backgrounds of this thesis, specifically showing the technical backgrounds and terminology of field-programmable technologies and (re-)configurable systems.

2.1 Field-Programmable Logic Devices

Firstly, traditional field-programmable technologies, which are also fundamental for my target dynamically reconfigurable processors, are described in this section. Although several kinds of field-programmable devices have been released since their introduction in 1980's, the most popularly-used Field-Programmable Gate Arrays (FPGAs) [1, 2] are focused in this section.

The following subsections briefly describe the basic technologies used to make FPGAs programmable. Finally, as an example of commercial FPGA devices, Xilinx Spartan-3A FPGA architecture is introduced.

2.1.1 Programmability

The FPGAs are composed of three fundamental components: logic blocks, IO blocks, and programmable routing. Circuits are implemented in an FPGA by programming each of the logic blocks to implement a small portion of the circuit logic. The programmable routing is programmed to make all the necessary connections between logic blocks. The IO blocks are configured as either an input pad or output pad.

The most popular technology to make FPGAs programmable is using SRAM cells to configure the programmable routing and the logic blocks. Fig. 2.1 shows three types of programmable switches used in SRAM-based FPGAs. In the case of the pass-transistor approach, the SRAM cell controls On/Off of the pass-gate. For the multiplexer approach, the RAM cells control which of the multiplexer's inputs should be connected to its output. This scheme would typically be used to optionally connect one of several wires to a single input of a logic block.

Most Xilinx and Altera FPGAs are SRAM-based [19, 20]. Alternative programmable technologies include anti-fuses and floating gate devices such as EPROM, EEPROM, and Flash which are

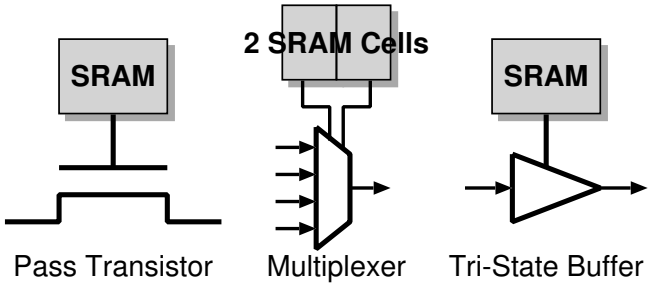


Fig. 2.1: SRAM-based Programmable Switches

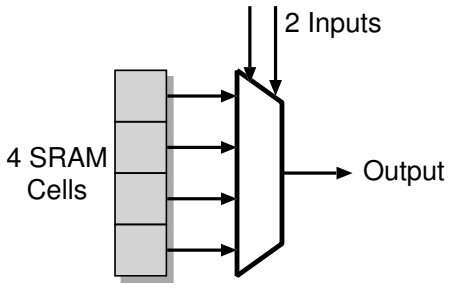
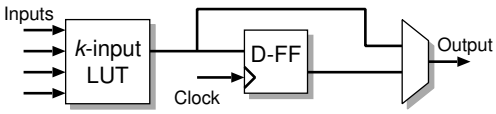
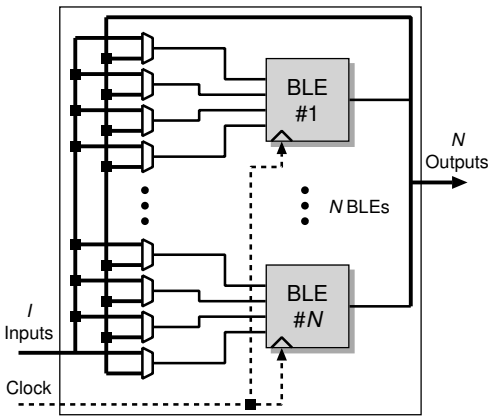


Fig. 2.2: 2-Input LUT



(a) Basic Logic Element (BLE)



(b) Logic Cluster

Fig. 2.3: Cluster-based Logic Block

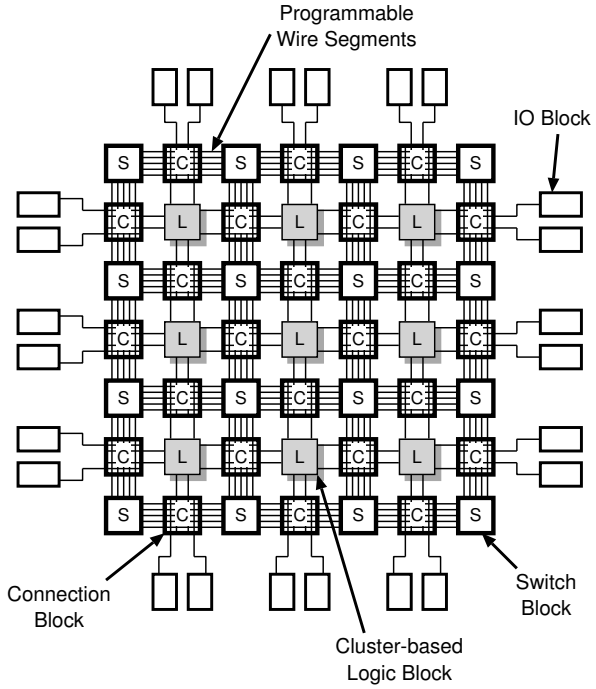


Fig. 2.4: Island-Style FPGA

mainly used in Complex Programmable Logic Devices (CPLD).

2.1.2 Cluster-based Logic Blocks

Most current commercial FPGAs use logic blocks based on Look-up Tables (LUTs). Fig. 2.2 shows an example of 2-input LUT which is implemented in an SRAM-based FPGA. A k-input LUT requires 2^k SRAM cells and a 2^k-input multiplexer. By programming the 2^k SRAM cells to be a truth table of a desired function, the k-input LUT can implement any function of k inputs.

For large LUTs with more inputs, most of the recent FPGAs use cluster-based logic blocks as shown in Fig. 2.3. The cluster-based logic block is a collection of basic logic elements (BLEs). The BLE consists of an LUT and a register, and the BLE output can be either registered or bypassed. The cluster-based logic block contains N BLEs and local routing to interconnect them. Ahmed

and Rose [21], taking deep sub-micron CMOS technologies into account, showed that the use of cluster sizes (N) between 3-10 and LUT sizes (k) of 4-6 will produce the best overall results for area-efficiency.

2.1.3 Routing Architectures

Fig. 2.4 depicts an FPGA with island-style interconnection structure which is the most popular among mesh-based FPGAs [22]. The island-style FPGA is composed of cluster-based logic blocks (L), switch blocks (S), connection blocks (C), and IO blocks. Each cluster-based logic block, which implements a user's logic, has inputs and outputs connected by the routing network. Between the rows and columns of an array of cluster-based logic blocks, routing channels are containing switch blocks and connection blocks. The connection blocks connect the input and output pins of a cluster-based logic block to the routing channel, and the switch block connects the wires of two intersecting (orthogonal) channels.

Because of recent advanced CMOS technologies, the delay of a circuit implemented in an FPGA is mostly due to routing delays rather than logic block delays. And also, most of an FPGA area is devoted to programmable routing. Hence, the requirement of investigating area-efficient FPGA routing architectures has been early recognized and accepted.

Brown and Rose [1] investigated the area-efficient FPGA architectures. They described some of the parameters of an FPGA routing architecture. The number of tracks or wires contained in a channel is denoted by W. The number of wires in each channel to which a logic block pin can connect is called the connection block flexibility F_c. The number of wires to which each incoming wire can connect in a switch block is called the switch block flexibility F_s. In [1], it was shown that, for the most area-efficient architectures, each switch block should contain 3 connections per wire, and each connection block should have a switch at 70-90% of all possible switch locations. These details are summarized by flexibility parameters $F_s = 3$ and $F_c = 0.7W$ to $0.9W$, respectively.

2.1.4 Xilinx Spartan-3A FPGA Architecture

As an example of commercial FPGAs, Xilinx Spartan-3A FPGA [23] is shown here. Fig. 2.5 depicts an overview of the Spartan-3A architecture. The Spartan-3 family FPGAs including Spartan-3A consist of five fundamental programmable functional elements:

- Configurable Logic Blocks (CLBs) array,
- Input/Output Blocks (IOBs),
- 18-Kb dual-port Block RAMs,
- 18-bit × 18-bit Multiplier Blocks, and
- Self-calibrating Digital Clock Manager (DCM) Blocks.

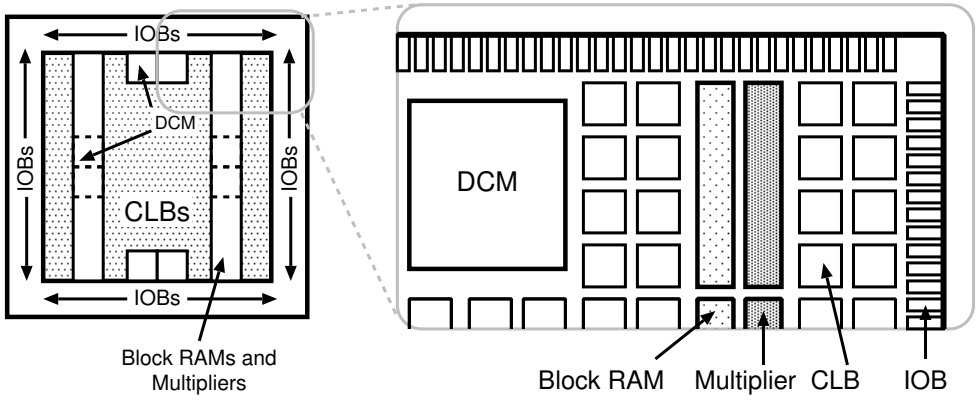


Fig. 2.5: Xilinx Spartan-3A Architecture Overview

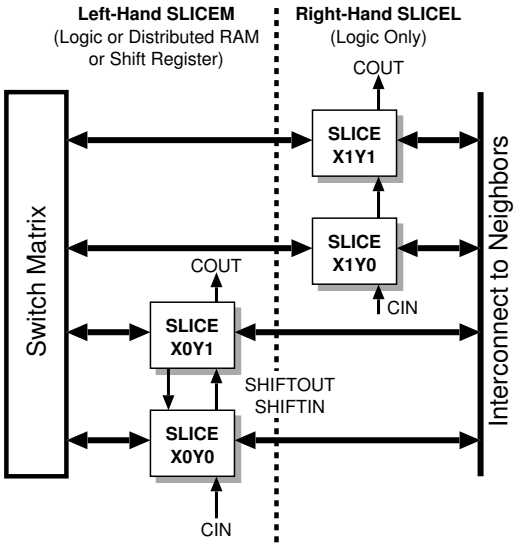


Fig. 2.6: Spartan-3 CLB Architecture

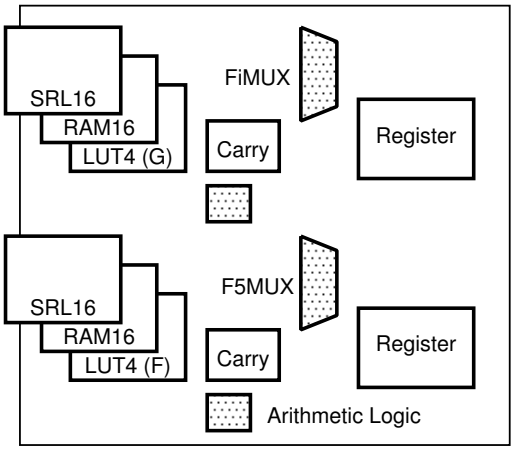


Fig. 2.7: Resources in a Slice (SLICEM)

The CLBs constitute the main logic resource for implementing synchronous as well as combinatorial circuits. As shown in Fig. 2.6, each CLB contains four slices, and each slice contains two LUTs to implement logic and two dedicated storage elements that can be used as flip-flops or latches. The LUTs can be used as a 16x1 memory (RAM16) or as a 16-bit shift register (SRL16), and additional multiplexers and carry logic simplify wide logic and arithmetic functions. Most general-purpose logic in a design is automatically mapped to the slice resources in the CLBs.

Each CLB comprises four interconnected slices, as shown in Fig. 2.6. These slices are grouped in pairs. Each pair is organized as a column with an independent carry chain. The left pair supports both logic and memory functions, and its slices are called SLICEM shown in Fig. 2.7. The right pair supports logic only, and its slices are called SLICEL. Therefore half the LUTs support both logic and memory (including both RAM16 and SRL16 shift registers) while half support logic only, and the two types alternate throughout the array columns. The SLICEL reduces the size of the CLB and

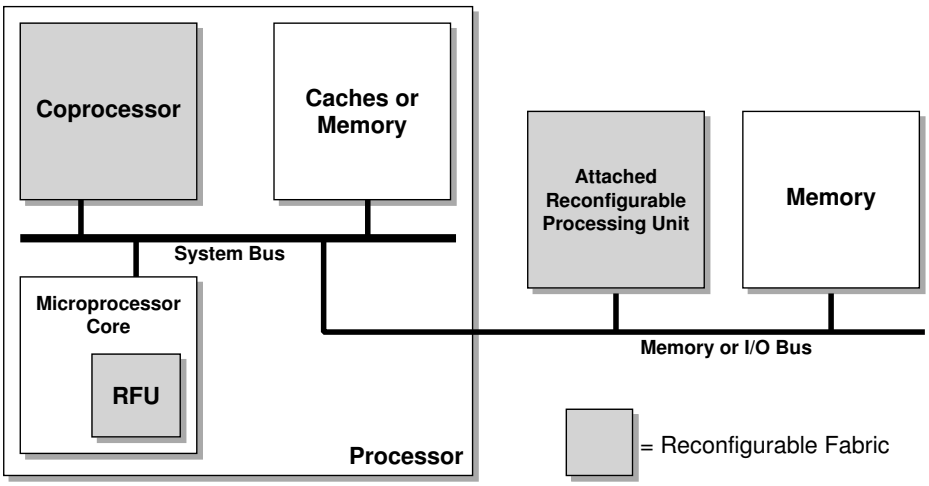


Fig. 2.8: Reconfigurable Fabric Integration into a System

lowers the cost of the device and can also provide a performance advantage over the SLICEM.

2.2 Reconfigurable Systems

As the next background of this thesis, reconfigurable systems are described in this section. The reconfigurable systems are computing systems that combine a reconfigurable fabric such as an FPGA with a microprocessor. These systems are fully exploiting the programmability or reconfigurability of the reconfigurable fabric, and they can achieve high performance of hardware execution with software-like flexibility [8].

In recent years, many reconfigurable systems have been released. Although it is difficult to clearly distinguish them in terms of architectures and device implementations, from the viewpoint of the coupling between a microprocessor and reconfigurable fabrics, as shown in Fig. 2.8, they can be classified into three groups [24]:

- an attached reconfigurable processing unit,
- a tightly coupled coprocessor, and
- a reconfigurable functional unit (RFU).

This section describes several conventional reconfigurable systems for each category. The dynamically reconfigurable processors focused in this thesis are expected to be tightly coupled with a microprocessor like a coprocessor or a reconfigurable functional unit.

2.2.1 Attached Reconfigurable Processing Unit

Attached reconfigurable processing units are integrated into a system on a memory or I/O bus. The systems with attached reconfigurable processing units, e.g. Splash 2 [25, 26], RM-I/II/III/IV/V [27,

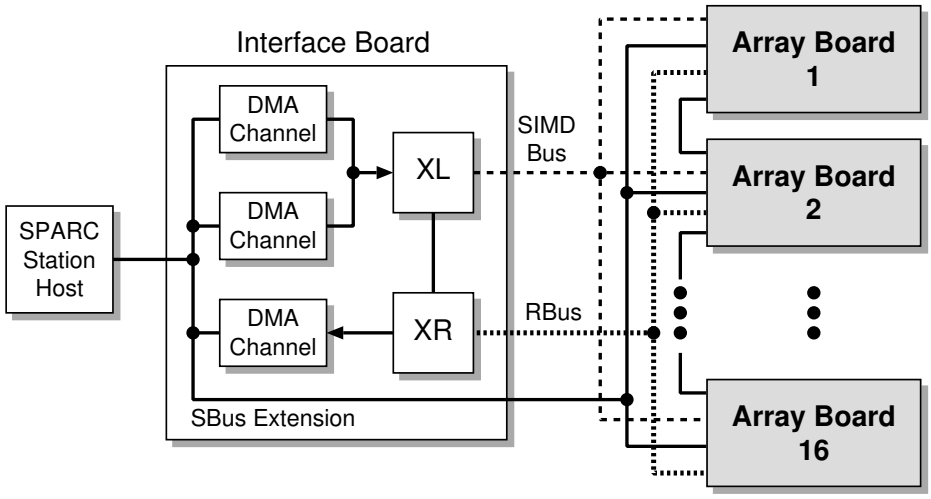


Fig. 2.9: Splash 2 System Architecture

28], RASH [29], and Teramac [30], have no direct access to the processor, and they are controlled over the bus. The primary feature of attached reconfigurable processing units is that they are easy to add to existing computer systems. However, due to the bandwidth and latency constraints imposed by the memory or I/O buses, they are not suitable for communication-intensive applications.

A representative example of the systems with an attached reconfigurable processing unit is Splash and Splash 2 from IDA Supercomputing Research Center (SRC)[1]. Splash 2 is an attached special-purpose parallel processor in which computing engines are FPGA devices. The architecture of Splash 2 is designed to accelerate solution of problems that exhibit at least modest amounts of temporal parallelism or data parallelism.

The Splash 2 consists of a Sun SPARC station host, an interface board, and from 1 to 16 array boards, as shown in Fig. 2.9. The interface board contains three bidirectional DMA channels capable of providing an aggregate bandwidth of 50MB/s. Two Xilinx FPGA devices, XL and XR, process incoming and outgoing data streams. The clock frequency is selectable by the host with 50-Hz interval step from 100Hz to 30MHz.

The array board contains 16 Xilinx XC4010 FPGAs (X1-X16) arranged in a linear systolic array and fully connected by a 16-way crossbar switch, which is regulated by the 17th FPGA (X0), as shown in Fig. 2.10. Although each of the FPGAs does not have on-chip local memory, it is connected to a 512-KB memory (M0-M16) on the array board. The memory is directly addressable by the host.

Each FPGA has 36-bit bidirectional datapaths to its left neighbor, to its right neighbor, and to the crossbar switch. The datapath to memory is 16-bit wide. The input to the array is provided by XL through the 36-bit SIMD bus to X0 of every board and X1 of the first board. Multiple array boards are linked together by extending the linear datapath from the right side (X16) of one board to the left side (X1) of the next. The crossbar switch is capable of storing up to eight separated and dynamically

[1]The SRC was renamed the Center for Computing Sciences in 1995.

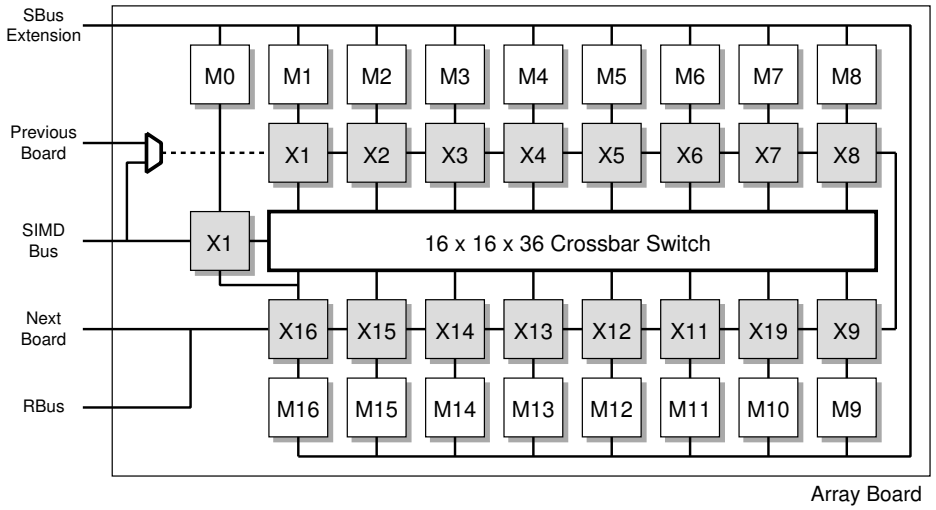


Fig. 2.10: Splash 2 Array Board

selectable configurations which specify a different set of connections among the 16 ports.

Applications for Splash 2 are developed by writing behavioral descriptions of algorithms in VHDL, which are then iteratively refined and debugged within the Splash 2 simulator. A wide variety of applications have been written for the Splash 2 system, including keyword and dictionary search, DNA and protein sequence analysis, and real-time image processing. By exploiting the linear systolic array structure, drastic performance improvement was achieved for the pipelined applications. On DNA sequence matching, Splash achieved over 300x the performance of a Cray-II supercomputer.

The Splash technology had been licensed to Annapolis Micro Systems, Inc. in 1994 with the commercial release of WILDFIRE^{TM}.

2.2.2 Tightly Coupled Coprocessor

Coprocessor systems consist of a microprocessor and reconfigurable fabrics. The reconfigurable fabric plays the role of a coprocessor which complements the microprocessor. In general, the coprocessor handles computation-intensive parts of a program, and the other parts are executed by the microprocessor. In the coprocessor systems, the reconfigurable fabric can have access to the same memory hierarchy to the microprocessor including several levels of caches, on-chip memories, and external memories. Therefore, there is a low-latency and high-bandwidth connection between the microprocessor and the reconfigurable fabric, which increases the number of stream-based functions that can profitably be run on the fabric. Examples of such systems include Garp [31, 32], Napa 1000 [33], PRISM [34, 35], and ArMen [36].

The Garp architecture, as shown in Fig. 2.11, includes a MIPS processor and a reconfigurable array of logic blocks. It combines the processor and the reconfigurable fabric on the same die. The reconfigurable fabric includes a two dimensional array of logic blocks. The logic blocks are similar to CLBs of the Xilinx 4000 series FPGAs. Four memory buses are placed vertically through the

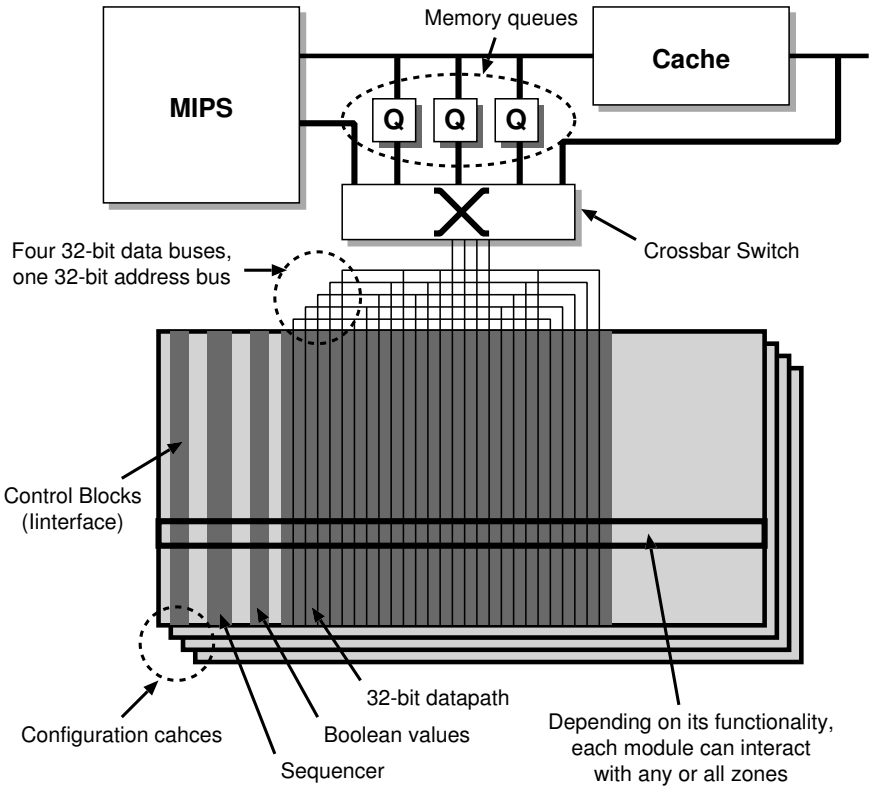


Fig. 2.11: Garp Architecture

rows. These buses are used to transfer data to/from the logic block array. They are also used for loading configuration data to the array and saving/restoring the array state. Another wire network is used for communications between the blocks. The MIPS processor is responsible for the loading and execution of the configurations. Several new instructions were added to the MIPS instructions set for these purposes.

2.2.3 Reconfigurable Functional Unit (RFU)

The tightest coupling between a microprocessor and a reconfigurable fabric occurs when the reconfigurable fabric is on the microprocessor's datapath, as in functional unit architectures like PRISC [37], Chimaera [38], and OneChip [39, 40]. All of these allow custom instructions to be executed. The reconfigurable fabric is on the processor datapath and has access to registers. However, these implementations restrict the applicability of the reconfigurable fabric by disallowing state to be stored in the fabric and in some cases by disallowing direct access to a memory, essentially eliminating their usefulness for stream-based processing.

The Chimaera consists of a host microprocessor augmented with a reconfigurable functional unit (RFU) as shown in Fig. 2.12. The RFU is given access to a host register file or a shadow register file. Up to nine input and one output operands can be used by an RFU instruction. Each RFU configuration determines by itself from which registers it reads its operands. Therefore, the

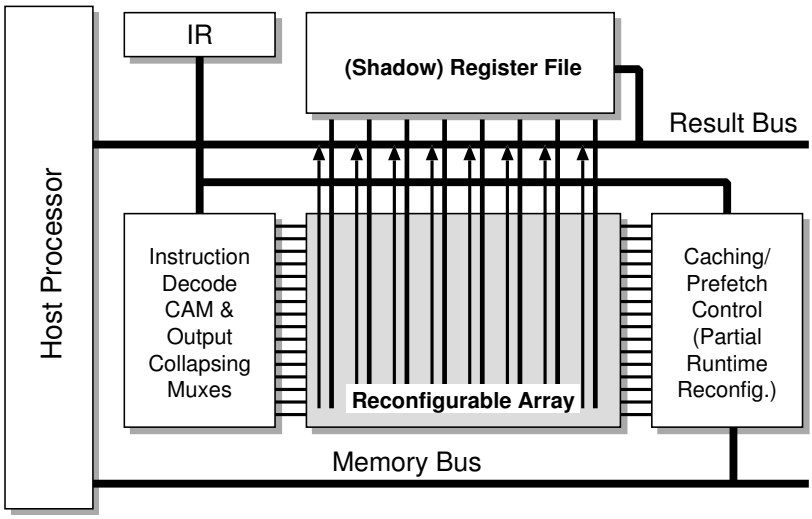


Fig. 2.12: Chimaera Architecture

RFU instruction format does not provide identifiers for the input operands. It includes only the RFUOP opcode, indicating that an RFU instruction is being called, an ID operand specifying which instruction to call and the destination register identifier. The programmable array is structured in rows of active logic between routing channels. A computing resource can use one or more rows. Multiple computing resources can be active on the RFU at a time.

2.3 Configurable Processors

Configurable processors give embedded system designers virtually unlimited choices in processor architectures, allowing them to customize several features to suit its application and design constraints at hand. For instance, designers can tune the processor's instruction set architecture (ISA) to the application's characteristics. Similarly, they can optimize the processor pipeline or datapath components for critical code segments in specific application domains. The resulting processors remain programmable and, in principle, can run any application, but they are optimized for a targeted application domain. Since configurability increases the design space, taking maximum advantage of this potential requires an efficient system for design space exploration.

While the idea of configurable processors is appealing, it opens up a new challenge: customization of an entire software tool chain including compilers, simulators, and debuggers. However, the software tools which handle the challenge are out of the scope of this thesis.

2.3.1 Tensilica Xtensa

Tensilica Xtensa [41, 42] is a processor core designed with ease of integration, customization, and extension in mind. Unlike common processors, Xtensa lets system designers select only the features required for a given application.

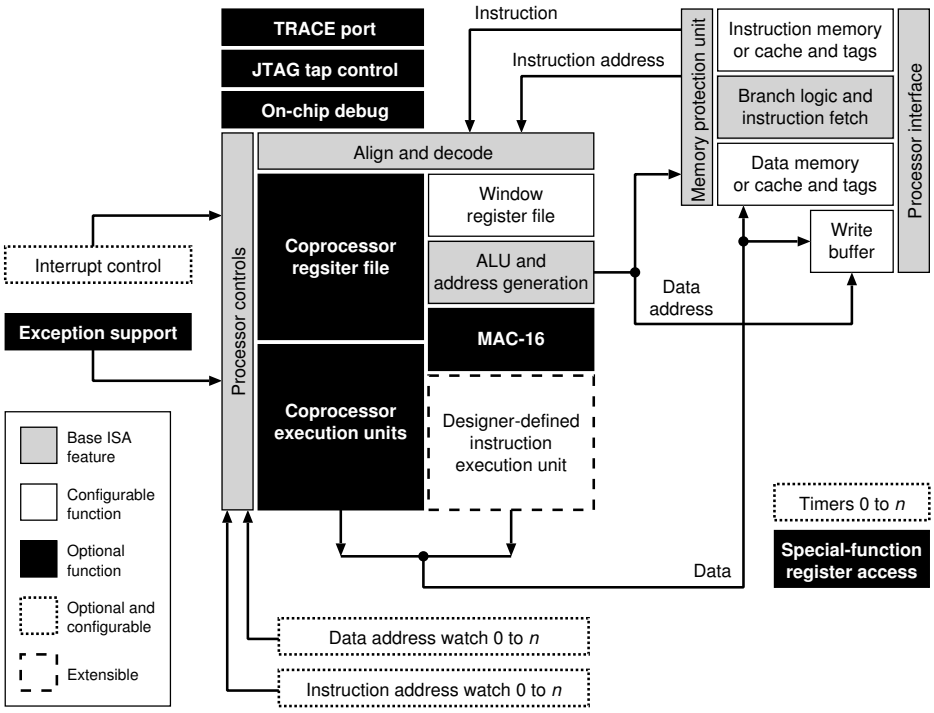


Fig. 2.13: Xtensa Architecture

The Xtensa ISA consists of a base set of instructions, which exist in all Xtensa implementations, plus a set of configurable options. The designer can choose, for example, to include a 16-bit multiply-accumulate option if it is beneficial to the application. The base ISA defines approximately 80 instructions and is a superset of traditional 32-bit RISC instruction sets.

Fig. 2.13 shows a high-level block diagram of Xtensa. The base ISA features correspond to roughly 80 instructions. Configurable options show designer choices, such as how many physical registers to include in the implementation, or the size of the instruction and data caches. Optional features are selections the designer can make, such as whether to include a 16-bit multiply-accumulate functional unit. Optional and configurable functions let the designer select whether to include data watchpoint registers and, if so, how many. Another configurable option is the designer-defined functional unit.

2.3.2 Toshiba MeP

Toshiba Media Embedded Processor (MeP) architecture [43,44] consists of a basic part and an extensible part. The basic configuration of the processor consists of the core instruction set, an instruction cache/RAM, a data cache/RAM, a direct-memory access (DMA) controller, and a bus bridge to a main bus, as shown in Fig. 2.14. The base processor is a 32-bit five-stage simple RISC core which consists of about 50K gates and can operate at 200MHz in 0.18μm CMOS technology.

In MeP, the following two very long instruction word (VLIW) extensions are supported:

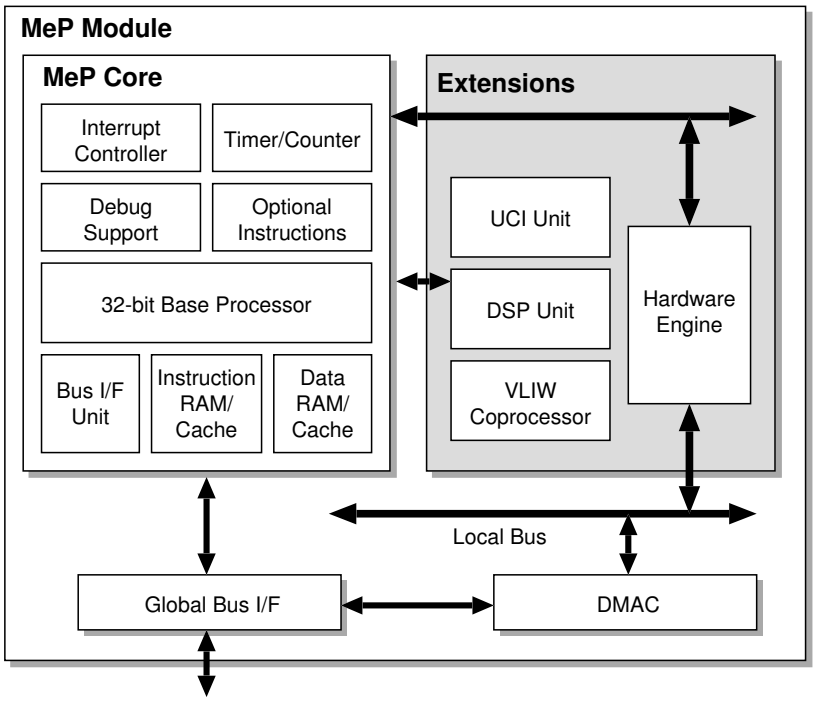


Fig. 2.14: MeP Architecture

- 2-issue VLIW execution unit with a 32-bit instruction length and
- 3-issue VLIW execution unit with a 64-bit instruction length.

In both cases, a core ALU and an extended coprocessor operate in parallel. In addition, load/store instructions to/from coprocessor registers support an auto-increment with the modulo addressing mode, in which general-purpose registers (GPRs) in the RISC core are used as address registers.

Each DSP extension instruction contains two register operands plus a 16-bit sub-opcode or immediate operand. Heavy iteration of a relatively simple task can be implemented as a hardware engine. A C or C++ function is mapped into a hardware engine. Both a control bus and a local bus are such that a load/store through the control bus, a DMA transfer over the local bus, and a hardware engine execution can operate in parallel.

Multiple processors can be connected to a shared memory using a global bus as shown in Fig. 2.14. In this context, each processor is referred to as a MeP module (MM). Each MM contains a basic processor configuration and may also contain a coprocessor, user-custom instructions, DSP extensions, hardware engines, and local memories. The architecture of each MM can be optimized independently based on its target application. Both 32- and 64-bit bus widths are supported.

2.3.3 Matsushita UniPhier

UniPhier [45, 46] platform-based SoC consists of five major units, UniPhier Processor, CPU, Stream I/O, AV I/O, and memory controller, as shown in Fig. 2.15. These five units are connected by a wide-

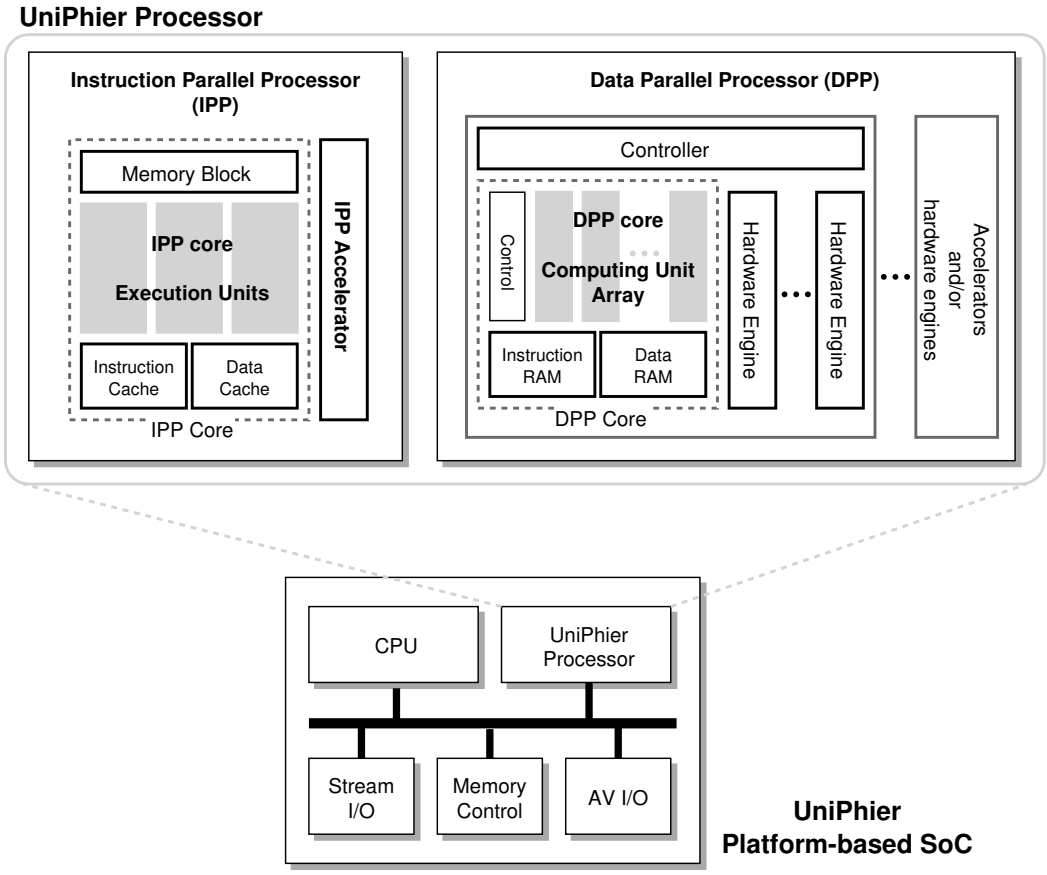


Fig. 2.15: UniPhier Architecture

range and high-bandwidth interconnect. The UniPhier platform can provide a suitable configuration of the UniPhier Processor for AV components and a common architecture for Stream I/O, AV I/O, and memory controller to customize for each target application.

The UniPhier platform has three types of components which exploit hierarchical parallelism. At first, an Instruction Parallel Processor (IPP) can be virtually operated as a multi-processor and is optimized for sequential and non-routine processes. The IPP can execute three instructions in parallel. Moreover, the IPP supports Virtual Multi-Processor (VMP) mechanism which is a parallel processing mechanism of multi-applications supported by hardware multi-threading. Secondly, a Data Parallel Processor (DPP) is a SIMD-style processor, which is optimized for parallel and non-routine processes. Finally, dedicated hardware engines assist the DPP for routine processes.

The SoC designers can configure these components to suit each application. With hierarchical parallelism to be exploited, UniPhier platform achieves flexibility, high performance, and low power consumption.

Chapter 3

Dynamically Reconfigurable Processors

This chapter describes dynamically reconfigurable processor architectures subjected to study in this thesis. At first, a background and an architectural overview of dynamically reconfigurable processors are introduced, and then several published architectures are reviewed.

3.1 Architecture Overview

As already introduced in Chapter 1, compared to application-specific integrated circuits (ASICs) or full-custom designs, FPGAs offer many advantages including reduced non-recurring engineering (NRE) cost and a shorter time to market. However, these advantages come at the cost of an increase in silicon area, an increase in power consumption, and a decrease in performance when designs are implemented on FPGAs. The existence of these inefficiencies in FPGA-based implementations is widely known and accepted [1, 2].

Kuon and Rose [47] measured the area, performance, and power gap between a 90-nm CMOS SRAM-programmable FPGA and a 90-nm CMOS standard-cell technology. Their evaluation results show that an FPGA is approximately 35 times larger and between 3.4 to 4.6 times slower on average than a standard-cell implementation. It is also observed that an FPGA consumes 14 times more dynamic power than an equivalent ASIC on average.

Recent dynamically reconfigurable processors attempt to overcome the disadvantages of FPGA-based reconfigurable systems and have the following three features:

- a coarse-grained processor array,
- dynamic reconfigurability, and
- a C-based programming methodology.

The combination of these features produces improvements in area- and power-efficiency with a shorter time to market compared to FPGAs. In the following subsections, each of the key features of the dynamically reconfigurable processors is described.

3.1.1 Coarse-Grained Processor Array

In traditional fine-grained FPGAs, primitive components are bit-level such as LUTs or AND-OR products. Due to the bit-level operations, operators for wide datapaths have to be composed of several bit-level processing units. This includes typically a large routing overhead for the interconnect between these units and leads to low silicon-area efficiency of FPGA-based reconfigurable systems. Also, programmable routing wires waste more power than hardwired connections. An additional drawback of the FPGAs is the high volume of configuration data needed for the large number of both processing units and routing resources. This implies a need for a large configuration data memory with its power consumption. The long configuration time, which is also implied by this problem, makes execution models relying on frequent changes of the configuration impossible.

The recent dynamically reconfigurable processors consist of two-dimensional arrays of coarse-grained processing elements (PEs), distributed memory elements, and IO elements. They are advantageous for area-efficiency by providing multiple-bit wide datapaths and word-level operators instead of bit-level reconfigurability. In contrast to the FPGAs, the wide datapath allows an efficient implementation of complex operators in silicon. Thus, the routing overhead caused by having to compose complex operators with bit-level processing units is avoided.

In general, each PE has a 4-bit to 32-bit ALU, a shifter, and a register file unit. Each unit is connected to each other with selectors which control a data stream. The array of many PEs connected with programmable routing resources like FPGAs provides various kinds of parallel datapaths such as Single Instruction and Multiple Data (SIMD), Multiple Instructions and Multiple Data (MIMD), and pipelining. Although the coarse-grained PE array has low flexibility compared to a fine-grained reconfigurable fabric, it can provide high performance and high area-efficiency for parallel multimedia applications. The PE with a small granularity such as 4-bit or 8-bit can generally improve flexibility with low datapath speed. In contrast, a large granularity can provide high area-efficiency and reduced configuration data only if the granularity suits the target application.

Regarding the interconnects between PEs, coarse-grained architectures also differ in several ways to FPGAs. The connections are multiple-bit wide, which implies a higher area usage for a single path. On the other hand, the number of PEs is typically several orders of magnitude lower than that of an FPGA. Thus, much fewer paths are needed, resulting in a globally lower area usage for routing. The lower number and higher granularity of communication paths allow also for communication resources, which would be quite inefficient for fine-grained architectures.

3.1.2 Dynamic Reconfigurability

Although the coarse-grained PE arrays actually provide high area-efficiency compared to fine-grained FPGAs, they are still inefficient compared to logically equivalent ASICs because of the programmability. A dynamic reconfiguration scheme is a primary technique to improve area- and power-efficiency of field-programmable logic devices like the coarse-grained PE arrays. A large amount

of configuration data can be stored into on-chip and/or off-chip memories, and the coarse-grained PE array can be dynamically reconfigured by reading out a desired configuration data from the memories. The dynamic reconfiguration offers a great advantage with respect to area- and power-efficiency because the PE array can be reconfigured so as to be optimized for various kinds of functions.

In the last few decades, fine-grained FPGA-based dynamically reconfigurable devices have been released. They include partially run-time reconfigurable FPGAs [48] and multi-context FPGAs [49, 50]. The fine-grained dynamically reconfigurable devices require a large amount of configuration data, and this implies millisecond-order reconfiguration time. Therefore, the dynamically reconfigurable FPGAs have been used for only restricted purposes such as a logic emulation system. Fortunately, the coarse-grained PE arrays can drastically reduce the need for configuration data, and they provide less than microsecond-order and practical dynamic reconfigurability.

In this thesis, the dynamic reconfiguration schemes are classified into two categories: a configuration delivery scheme and a multi-context scheme.

(1) Configuration Delivery Scheme

The *configuration delivery scheme* is used for the systems in which the dynamic reconfiguration is not performed frequently. In this method, as shown in Fig. 3.1, the configuration data for each reconfigurable element including PEs and switching resources is stored in on-chip configuration data memory modules. The configuration data is delivered to corresponding elements in sequence via a dedicated configuration bus. Since the configuration data memory modules are centralized on the chip, the configuration delivery scheme has an advantage of area-efficiency, and communication with outside of the chip is easy to be done. The reconfiguration time for the configuration delivery scheme is almost microsecond order. This is preferable to FPGAs with millisecond-order reconfiguration time, but operations of the PE array have to be suspended during the reconfiguration in most cases.

(2) Multi-Context Scheme

The other dynamic reconfiguration method, the *multi-context scheme*, is supported in recent dynamically reconfigurable processors. In this method, the configuration data corresponding to a PE-array datapath circuit of the PE array is called *context*, and each context contains operational instructions of each PE and intra/inter-PE connection instructions. Each reconfigurable element, as shown in Fig. 3.2, provides its own *context memory* which stores a set of contexts. A context number is delivered to all of the reconfigurable elements and used as a pointer to the context memories. By changing the context number and reading the context memory simultaneously, all the reconfigurable elements can switch the context in parallel with only one clock cycle. This method implies that the configuration data corresponding to a context is distributed to each reconfigurable element, and a context is switched by reading out the configuration data from each of the context memories.

There are different context switching control methods for the multi-context dynamically recon-

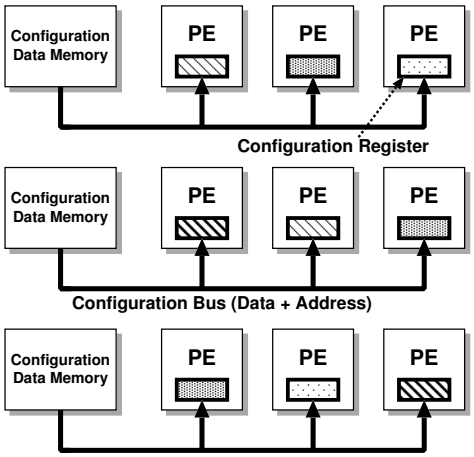


Fig. 3.1: Configuration Delivery Scheme

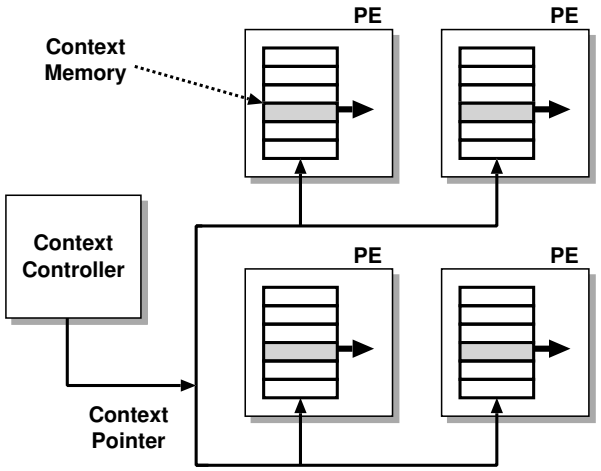


Fig. 3.2: Multi-Context Scheme

figurable devices. The three methods supported by recent architectures include data-driven control, state transition control, and program counter-based control. At first, Ling and Amano [51] proposed the data-driven control method in their architecture WASMII. In WASMII, a context is activated when all input data are present. Secondly, in the state transition control method, a simple controller has a state transition table which contains statically defined context switching patterns. Finally, the program counter-based control is the simplest method based on an incrementing counter. As similar to commonly used microprocessors, the controller has a counter for the context number, and the context indicated by the counter is activated.

Although the data-driven method provides dynamic and flexible context switching controls, it has a design difficulty because of the hardware overhead for detecting an activated context [52]. Therefore, the simpler methods including the state transition control and the program counter-based control mechanisms are commonly used in recent multi-context dynamically reconfigurable processors. Since the proposal of WASMII in 1992, our project has investigated multi-context dynamically reconfigurable processor architectures, and we focus on them also in this thesis. And also, we assume the state transition control and the program counter-based control methods for the context switching control because of their generality.

3.1.3 C-based Programming Methodology

Due to their field-programmability, the dynamically reconfigurable processors can execute various kinds of parallel algorithms. For a stream-based processing like image processing, streaming data are distributed into memory modules and executed by the PE array in data-parallel and SIMD fashion. Moreover, since the PE array has many independent operators such as shifters, ALUs, and registers, and switches a context in a cycle-by-cycle manner, it can perform like a Very Long Instruction Word (VLIW) processor exploiting instruction-level parallelism (ILP).

For dynamically reconfigurable processors, a C/C++-based programming methodology is com-

monly accepted [53]. Although basic C language is suitable to describe algorithmic behaviors, it's not suitable for data flow or continuous computation [54]. In order to apply C language to describe the hardware behavior for ASICs and FPGAs, several C-based hardware description languages and high-level synthesis technologies have been proposed. They support extended features such as timing and communications. Fortunately, since the coarse-grained PE array consists of ALUs, shifters, and register files, an arbitrary data flow graph described in a C-based language can be efficiently mapped onto the PE array.

3.2 Time-multiplexed Execution Models

As mentioned above, in the multi-context dynamically reconfigurable processors, each reconfigurable element has its own context memory for storing multiple hardware contexts. A computational task is generally divided into multiple contexts under the context memory capacity constraint and the PE array executes the task rapidly switching the contexts in a cycle-by-cycle manner. In this thesis, the execution method for a single computational task divided into contexts is called *context-level time-multiplexed execution*.

In the context-level time-multiplexed execution, various kinds of datapaths or logic functions can be virtually executed on the time-multiplexed PE array. The feature can be expected to dramatically improve area- and power-efficiency of the PE array. On the other hand, the area of the context memories and the delay for the context switching control are significant overheads. Thus, it can turn out to be harmful for the area-efficiency with a high degree of time-multiplexing.

The PE-array size which is defined by the number of available PEs in a context is the most remarkable factor that affects the degree of time-multiplexing (i.e., the number of required contexts) for each application. In this thesis, it is assumed that the PE array with a certain size has a certain number of multipliers and distributed memory modules. This assumption implies that the PE-array size linearly increases with the number of available PEs. If a large PE-array size is taken, area- and power-efficiency can be degraded though the performance can be improved by the parallel processing with many PEs. Meanwhile, a small PE-array size can cause the increase of required contexts and execution clock cycles with the improvement of area- and power-efficiency. This trade-off is closely investigated in Chapter 4.

The dynamically reconfigurable processor generally aims for an application group or domain, not for a single particular application. Moreover, for a large-scale application over a context memory capacity, it is required to be divided into smaller child tasks, which are executed sequentially. Then, it's not practical to equip large context memories on the PE array so as to hold all of the computational tasks. Actually, a large configuration data memory is integrated outside the PE array core for backup of many computational tasks. The configuration data corresponding to a requested computational task is transferred to each context memory independently of PE-array operations, and then a task controller dispatches the task on-demand. In this thesis, we refer to this execution method as *task-*

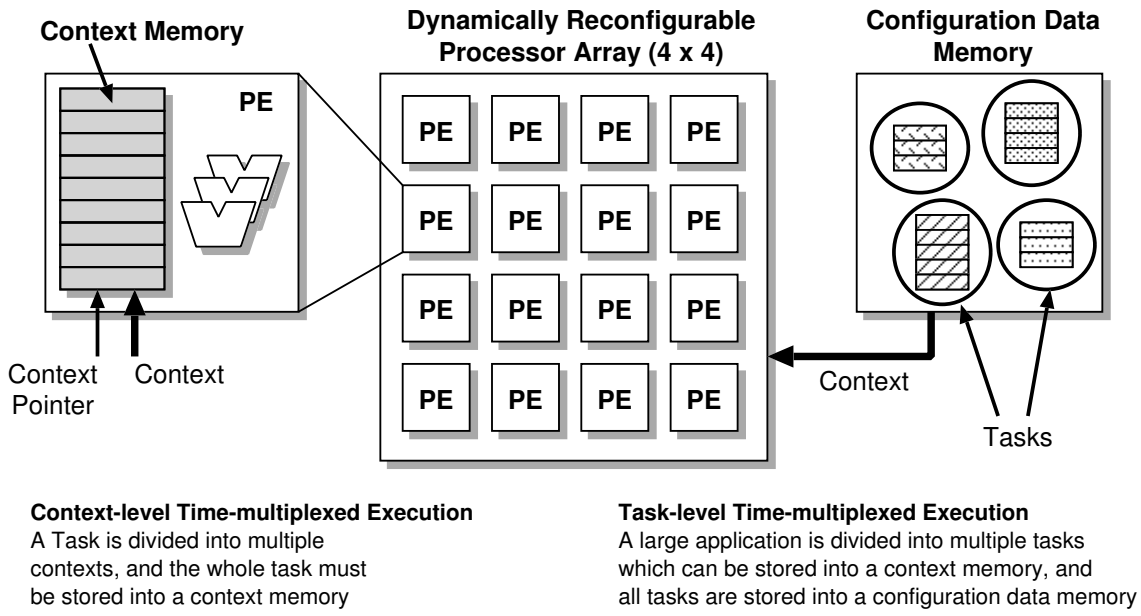


Fig. 3.3: Hierarchical Time-multiplexed Execution Models in This Thesis

level time-multiplexed execution.

It is guessed to be preferable that computational tasks to be executed on the single dynamically reconfigurable processor core have relatively similar algorithmic characteristics. This is because the dynamically reconfigurable processor core should be optimized to suit a target application domain. The task-level time-multiplexed execution method leads to support more and more applications and improve area- and power-efficiency much further. However, it can cause a microsecond-order time overhead for switching the computational tasks. If frequent task switches can be expected, a detailed trade-off analysis taking the overhead into account is required.

Fig. 3.3 shows the concept of two time-multiplexed execution models investigated in this thesis. Throughout the detailed analysis of the time-multiplexed execution models, we aim to suggest a guideline to design efficient architectures for a target application. In addition, we develop various application examples as case studies and demonstrate the potential of the dynamically reconfigurable processors for embedded products.

3.3 Review of Published Architectures

This section reviews several dynamically reconfigurable processor architectures released by consumer electronics makers in the last decade. They include not only commercially available products but also new devices under research and development. Academically proposed architectures including PARS [55], MorphoSys [56], and WPPA [57] are not described in this section because of space limitation, but Section 3.4 contains only their architectural summaries. In addition, architectures

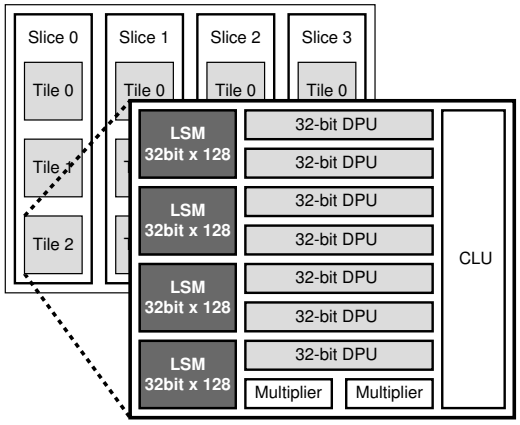


Fig. 3.4: Chameleon CS2112 Reconfigurable Processing Fabric

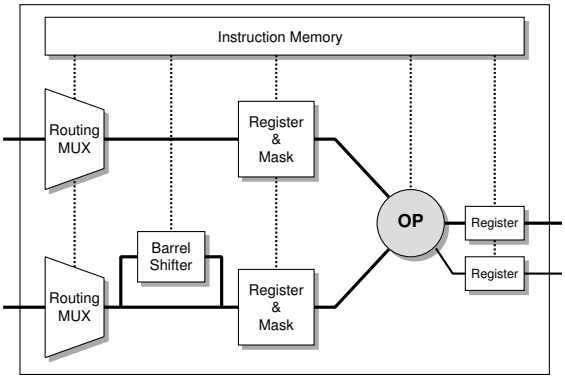


Fig. 3.5: Chameleon CS2112 Datapath Unit (DPU)

which have not been published in detail, such as SONY Virtual Mobile Engine (VME)[1] [58, 59], are also omitted. This section also does not contain NTT's dynamically reconfigurable devices, PCA-1 [60] and PCA-2 [61], because their architectures based on asynchronous circuits, serial data communication, and fine-grained logic elements are completely different from others.

3.3.1 Chameleon CS2112

The Chameleon's Reconfigurable Communication Processor (RCP) [62] called CS2112 [2], comprises a Reconfigurable Processing Fabric (RPF), programmable input and output (PIO) banks, and an embedded microprocessor. As shown in Fig. 3.4, the top-level of the RPF is divided into four slices, each of which can be reconfigured independently of the other slices. Each slice is subdivided into three tiles, and each tile consists of seven 32-bit datapath units (DPU), two multipliers (MPU), four local-store memories (LSM), and a Control Logic Unit (CLU).

The DPU shown in Fig. 3.5 is a fundamental computational element in the fabric. It has a basic word length of 32 bits but has special operations that allow it to operate in SIMD fashion on four 8-bit data streams or two 16-bit data streams. The core of the DPU is the 32-bit Operator (OP) that performs arithmetic and logical operations. The MPUs can perform 16×24-bit or 16×16-bit single-cycle multiplications. The LSM is a multi-ported 32-bit $\times$ 128-word RAM. Each DPU is programmed with eight user-definable instructions stored in its instruction memory (context memory). Each instruction specifies a complete configuration for the DPU. The CLU directly implements a finite-state machine to select the DPU and MPU instructions stored in the instruction memory. Nearby DPUs are connected to each other in a full crossbar connection. More distant DPUs are connected with routing with a one-clock pipeline delay.

[1] In spite of its undisclosed architecture, the VME has been early integrated into portable audio players and game machines including Play Station Portable (PSP).

[2] The technologies of Chameleon's reconfigurable processors are now licensed by Intel Corporation.

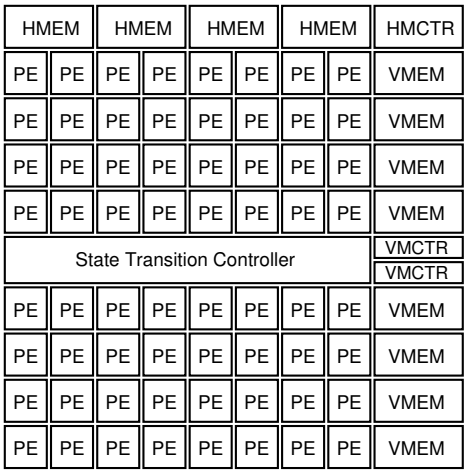


Fig. 3.6: DRP Tile Architecture

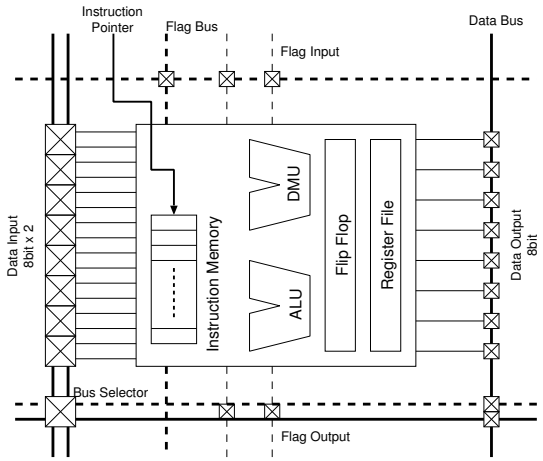


Fig. 3.7: DRP PE Architecture

The RCP consists of a configuration controller and two configuration planes (contexts). The active configuration actually controls the fabric, and the background plane is used to hold another configuration. The controller can load a new configuration data into the background plane while the fabric is running with the active plane. To program all four slices, a new configuration data can be loaded in less than 3 μsec. The entire system can be reconfigured in one clock cycle by switching the configuration data from the background plane to the active plane.

The RCP has been implemented at a 0.25 μm CMOS technology with a 125-MHz clock. From the performance evaluation results with FIR filter and FFT, the CS2112 reconfigurable processor can perform about 6 times faster than the equivalent Texas Instrument's processor.

3.3.2 NEC/NEC Electronics DRP-1

Dynamically Reconfigurable Processor (DRP) is a coarse-grained reconfigurable processor that was released by NEC Electronics in 2002 [63]. The basic building unit of the DRP architecture is a Tile shown in Fig. 3.6. Each Tile consists of 64 PEs, 8 vertical memory modules (VMEMs) with 2 controller (VMCTRs), 4 horizontal memory modules (HMEMs) with one controller (HMCTR), and a state transition controller (STC). By combining multiple Tiles, designers can construct a DRP core with a desirable size.

Fig. 3.7 shows the PE architecture which is composed of an Arithmetic Logic Unit (ALU), a Data Manipulation Unit (DMU), a Flip-Flop Unit (FFU), and a Register File Unit (RFU). The ALU performs addition, subtraction and so on. The DMU is a logic operator, and it performs the combinations of logical AND, OR, and shift. The FFU is an 8-bit wide flip-flop unit, and the RFU is an 8-bit wide and 16-entry deep register file.

In a DRP core, memory modules are laid-out in jail-bar like structure. VMEMs separate the tile columns, and HMEMs take the top and bottom. Each of VMEM modules is an 8-bit $\times$ 256-word memory with one read port and one write port. It can be also configured as a FIFO storage. Each of

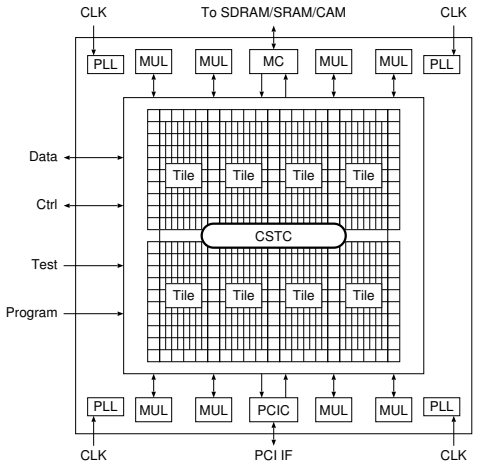


Fig. 3.8: DRP-1 Architecture

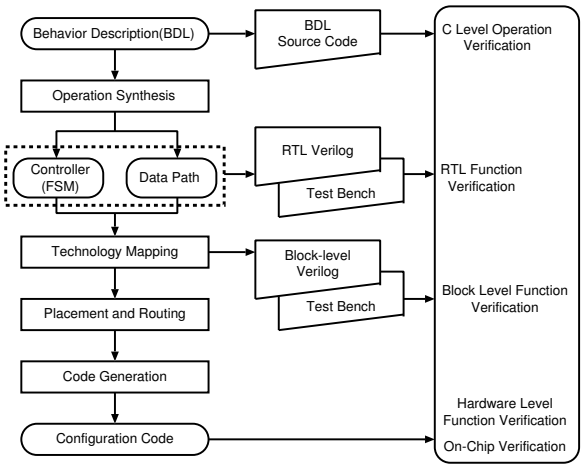


Fig. 3.9: Programming Flow of DRP

HMEM modules is an 8-bit × 8192-word memory with one read or write port. Together with PE's FFU and RFU, the DRP supplies a uniform distribution of memory resources.

Each of the reconfigurable elements such as PEs has its instruction memory (context memory), in which 16 contexts can be stored. According to a context pointer delivered from the STC, all of the reconfigurable units read the corresponding context and then are reconfigured. The STC is a programmable sequencer which carries out context switching based on a simple state transition, and can store up to 64 states. Each state is associated with the context pointer which indicates a particular context. The STC can receive event signals from the PE array to take a branch conditionally. The maximum number of branches that can be specified from the PE array is four.

Fig. 3.8 depicts the prototype chip DRP-1, which is composed of a DRP core with 4×2 Tiles. It has been fabricated with a 0.15-μm CMOS technology. It consists of 8 Tiles, eight 32-bit multipliers, an external SRAM controller, a PCI interface, and 256-bit I/Os. The maximum operational frequency is 100MHz. Although the DRP-1 is used as an attached reconfigurable processing unit, the DRP core can be used as an IP core on SoCs with an embedded processor. In this case, the number of Tiles can be chosen so as to achieve required performance with minimum area.

The integrated design environment called Musketeer is provided for the DRP-1. The application design flow is drawn in Fig. 3.9. It includes a high-level synthesizer, a technology mapper, simulators, and a layout/viewer tool. Each synthesis phase is seamlessly integrated. DRP programs can be written in the C-based hardware description language called Behavioral Design Language (BDL), synthesized, and mapped directly onto the DRP-1.

3.3.3 IPFlex DAPDNA-2

DAPDNA [64] is a dynamically reconfigurable processor architecture released by IPFlex in 2002. In DAPDNA, sequential processing and large-scale data processing are distinguished and executed in parallel. The sequential processing is performed by a 32-bit RISC processor core called Dig-

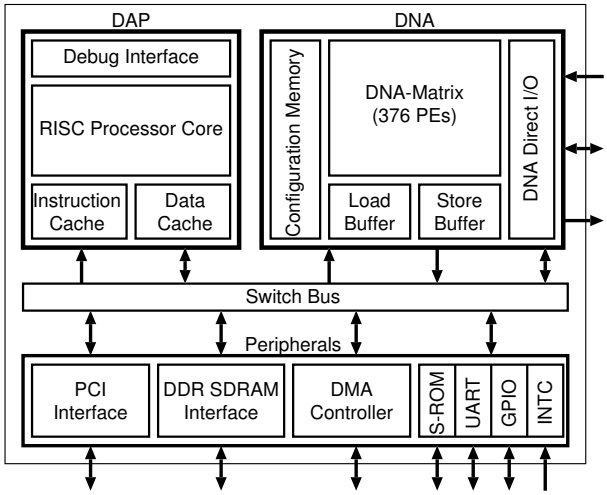


Fig. 3.10: DAPDNA-2 Architecture

Fig. 3.11: Types of DAPDNA-2 PE

PE	Qty	Function
EXE	168	Arithmetic and logical Ops.
DLY	138	Variable delay for data sync.
RAM	32	Temporary memory
C16E	12	Address generator
C32E	12	Address generator
LDB	4	Data Input from processor bus
STB	4	Data Output to processor bus
LDX	4	Data Input from Direct I/O
STX	4	Data Output to Direct I/O

ital Application Processor (DAP). And, the large-scale data processing is accelerated with a two-dimensional PE array called Distributed Network Architecture (DNA) matrix. The DNA matrix can dynamically reconfigure its datapath and exploit loop- and data-level parallelism.

Fig. 3.10 depicts DAPDNA-2 architecture which is the second generation processor of DAPDNA architecture. It consists of a DAP core and a DNA-matrix. The DAP core has an independent 8-KB of instruction cache, and 8-KB of data cache. The DNA matrix has 32-bit processing elements (PEs) with multi-context dynamic reconfigurability. The operating frequency of each processor core is up to 166MHz. The DAP core performs system-control and general-purpose operations. The DNA-Matrix provides capability for high speed data processing and scalability, with parallel computing and dynamic reconfigurability, respectively. This tightly coupled dual-core architecture realizes both flexibility and high performance.

The DNA-Matrix includes several types of processing elements as shown in Table 3.11. To map an algorithmic data flow graph directly onto the hardware, several types of PEs are placed in a two-dimensional PE array consisting of 8×8 PEs. This 8×8 unit is considered to be a segment, and the DNA-Matrix includes 6 segments, the total number of PEs being 376. There are two major categories of PEs. The first category is data processing where the combination of EXE/DLE/RAM has a complete capability to execute various operations on a data stream. The second is data I/O, and it is possible to generate data streams autonomously.

The DNA-Matrix has rich bus structure to realize various types of algorithms, and the DNA-Matrix Bus consists of a column-wise vertical bus which includes 8×32-bit wires corresponding to the row number. The DNA-Matrix Bus covers all rows in a segment, and the source PE can output its signal to all columns in a segment. Therefore, connections between arbitrary PEs are available, regardless of the relative position between source and destination PEs in a segment. The segment boundary connections are restricted slightly to realize a fixed operating frequency. The boundary

PEs generate their output at the position of boundary PEs within the neighboring segment.

Configuration data of the DNA-Matrix is in two orders of magnitude smaller than fine-grained FPGAs. With this configuration size, the DAPDNA processor can hold entire DNA configuration data within context memories of each PEs. The DAPDNA-2 holds four contexts in the context memories: one in the foreground and three in the background. The proximity of configuration memory enables the DNA-Matrix to switch configuration in one clock cycle. The trigger of a dynamic reconfiguration event can be generated by two sources: the control signal from DAP core and reconfiguration signal from the DNA-Matrix itself. The total reconfiguration time using the DAP core is usually less than 1μs.

The result of performance evaluations shows the DAPDNA-2 performs more than 28,000 million instructions per second (MIPS) at a clock speed of 166MHz and is capable of more than 9,000 million MACs per second (MMACS) of multiply-accumulate operations with 16-bit inputs and a 32-bit output. And, the DAPDNA-2 can execute image filtering operations such as a Laplacian filter at 1 giga pixels per second. The same operation on a Pentium 4 (3.06 GHz) is topped at 16 mega pixels per seconds and this implies the DAPDNA-2 has 63-times performance gain.

In recent years, IPFlex continues to release enriched DAPDNA-2-based architectures. DAPDNA-3A [65] is based on multi-core structure, and consists of 4 DAPs and 4 DNA-Matrices. Each DAP and DNA-Matrix are connected by a reconfigurable interconnection called AXION. Furthermore, the DNA-matrix in DAPDNA-IMS and DAPDNA-IMX [66] is enlarged to 955 16-bit PE array for the special purpose of image processing.

3.3.4 Hitachi FE-GA

Hitachi proposed Flexible Engine/Generic ALU Array (FE-GA) in 2005 [67]. The FE-GA is implemented using TSMC 90-nm Low Power CMOS technology, and it operates at 266MHz with power consumption of 200mW and with an area of 5mm^2.

The FE-GA consists of 24 ALU cells, 8 MLT cells, 10 load/store (LS) cells with a local RAM, a sequential manager (SEQM), a configuration manager (CFGM), a crossbar switch cell (XB), and a system bus interface as shown in Fig. 3.12. Each computing cell (ALU and MLT cells) is connected to its neighboring four cells. Therefore, the length of wires can be extremely short, making for a short delay that provides a high clock frequency. In addition, each of the LS cells, which are interface cells for local RAMs, can be connected to any computing cell on the utmost-left or utmost-right sides via XB connection.

The configuration data for each cell is managed by the CFGM and transferred to configuration registers (i.e., context memories) of each cell in the background. This configuration data is referenced by a pointer so that it can be compressed to 20% of its original size on average. The context switch of each cell is controlled by the SEQM, which generates a switching trigger using its built-in counters or the computation result of counters or comparisons of ALU cells. The SEQM can consider a

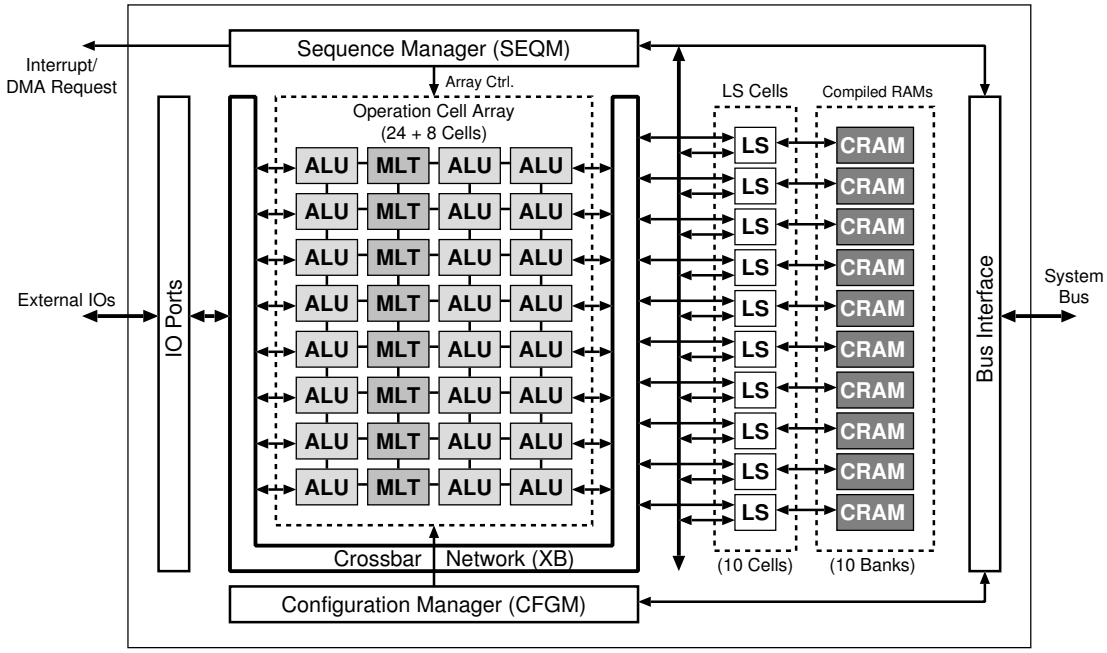


Fig. 3.12: FE-GA Architecture

conditional branch for the trigger, so the FE-GA can perform run-time context switches flexibly.

Fig. 3.13 shows an ALU architecture of the FE-GA. Each ALU cell consists of a general 16-bit ALU, a shifter (SFT), a data through logic circuit (THR), and an input delay logic circuit. All components of the ALU cells can be operated simultaneously. The data through logic circuit transfers the data from a cell's output to its neighboring cell's input without blocking any ALU operation. In addition, some ALU cells can send their result to the SEQM as a switching trigger that switches the context of all the cells by the result.

Each MLT cell consists of, as similar to an ALU cell, a general multiplier, a data through logic circuit, and an input delay logic circuit. All components of the MLT cells can be operated at the same time as ALU cells. Each LS cell shown in Fig. 3.14 consists of a port (A0/A1) connected to the XB, a port (B) connected to the bus interface and a port (0/1) connected to a dual-port RAM (local RAM). Because the LS cells can generate load and store addresses without using the ALU cells, the ALU cells can be saved for user applications.

Evaluation results based on TSMC 90-nm LP CMOS technology demonstrates that the operational frequency is 266MHz in worst cases, and the performance is 17.0 GOPS for 16-bit integer operations. The estimated area of the FE-GA is 5.0mm^2 including 40-KB local RAM, and power consumption is estimated as 210mW. In addition, several signal processing applications such as FIR filters and DCT have been implemented on the FE-GA, and it can perform at about 10 times fewer clock cycles than a DSP.

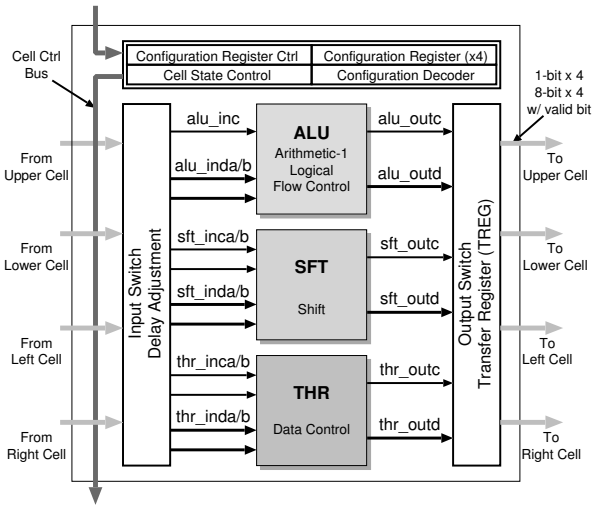


Fig. 3.13: FE-GA ALU Cell Architecture

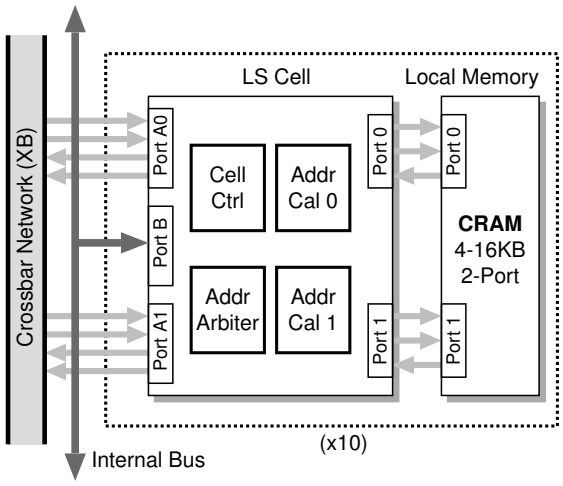


Fig. 3.14: FE-GA LS Cell Operation

3.3.5 IMEC ADRES

IMEC ADRES [68] has two functional views as shown in Fig. 3.15: a VLIW view and an accelerator view as an array of reconfigurable cells (RCs). While the VLIW processor is optimized for control and load/store operations, the accelerator is optimized for data-parallel processings. In acceleration mode, the functional units of the VLIW form the first row of the array. Orthogonal buses facilitate data transport within the array. The VLIW part includes up to eight functional units organized in a row. The units communicate with each other through a horizontal data bus for data exchange. A part of the units can communicate vertically with a common register file for data load and store.

The attached array is composed of several rows of RCs organized in a matrix form. The behavior of each RC is controlled by a locally stored set of several contexts. During the execution phase, a central controller allows dynamic reconfiguration of the RCs within a cycle. Each RC, as shown in Fig. 3.16, includes a local context memory (denoted by configuration RAM in the figure), an ALU, input and output multiplexers, and a register file for local data storage.

The data exchange between RCs is done with orthogonal interconnect-networks. There are two levels of interconnect for internal data exchange between the units: a global bus for each row or column spans the entire array. Additionally, the array is subdivided into four segments. Within a segment, a local interconnect is provided such that a RC can get input data directly from each of its horizontal or vertical neighbors.

The RC has been implemented with the Infineon Technologies CMOS 130nm 6 metals process technology. A context memory is designed with a 40-bit $\times$ 32-word SRAM. The schematics show that one RC requires 63266 transistor counts, and the total area is 0.196 mm^2. The area breakdown of the RC components is as follows: 50% for the context memory, 6% for external interfaces for input and output, 9% for a register file, and 19% for an ALU. About 15% of the area of an RC are consumed by the interconnect between the components.

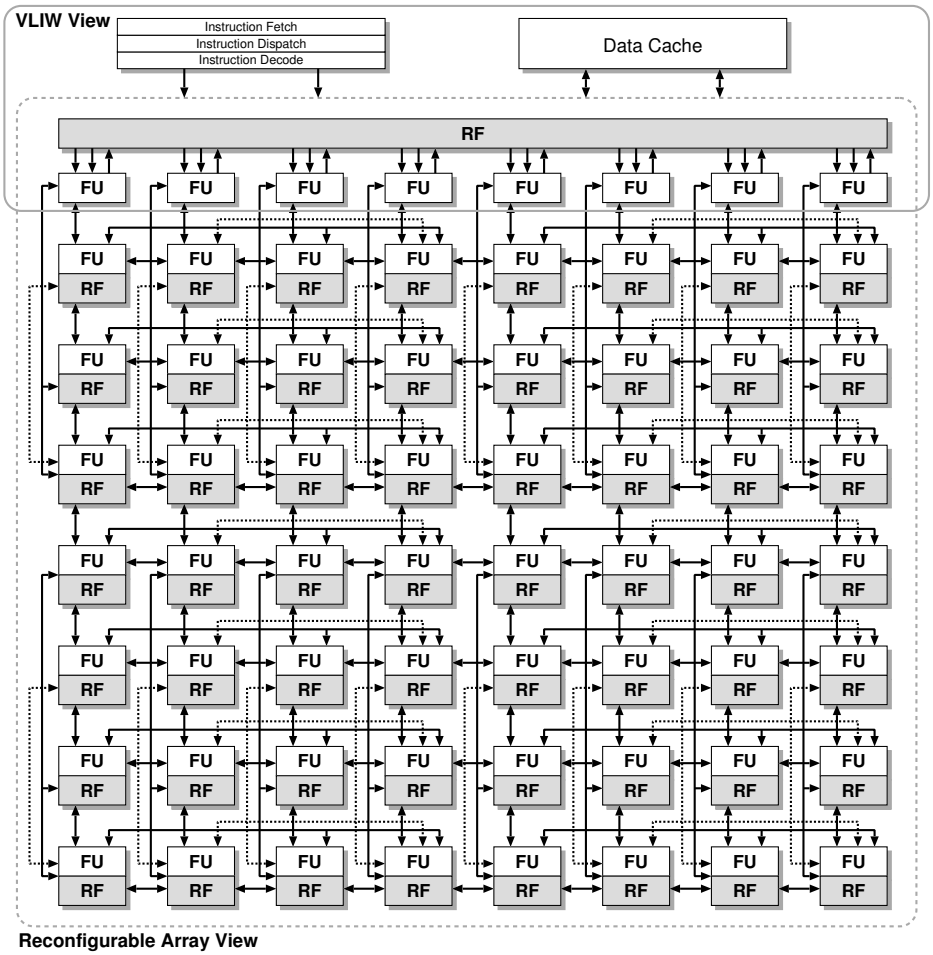


Fig. 3.15: ADRES Core Architecture

The maximum operational frequency of the RC is 60MHz or 16-ns cycle time. At a target frequency of 100 MHz, the power consumption for one RC is about 1.7mW. For performance evaluations, an H.264/AVC video decoder has been implemented on the ADRES architecture. The evaluation results show that the decoding cycle for the array view is about 88% faster than the VLIW view. The required clock frequency is 184MHz in the VLIW view and 98 MHz in the array view. The average power consumption at 100MHz would be around 46.3mW.

3.3.6 Fujitsu Cluster Architecture

Fujitsu has proposed a coarse-grained multi-context dynamically reconfigurable processor suitable for wireless communication [69]. The architecture consists of a hierarchical PE array called a cluster.

Fig. 3.17 shows a block diagram of the cluster. It contains several kinds of PEs, inter-PE networks, and a sequencer. To increase the cluster's area-efficiency, heterogeneous PEs are used in a cluster. The PE can be a 16-bit ALU, a MAC, a selector, a shifter, an address generator (counter), a one-port memory, a two-port memory, a register file, and variable delay line. The ALU supports

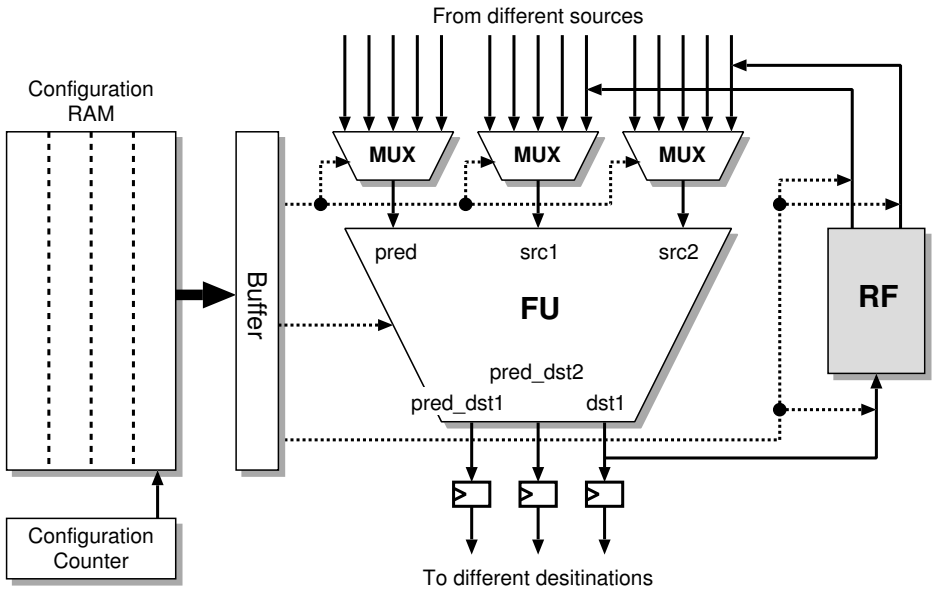


Fig. 3.16: ADRES Reconfigurable Cell

logical, arithmetic, compare, and shift instructions, similar to DSPs. The "compare" instruction generates a signal that is used as a select signal by the selector and/or a transition signal by the sequencer. The MACs support 16×16 bit multiplication and accumulation.

The topology of inter-PE networks is a kind of indirect three-cube network and consists of three selector levels. The number of logical steps is the same in all paths. In our first implementation, the network corresponding to a 64×64 switch that consists of 4×4 switches. The sequencer controls the dynamic reconfiguration of the PE and inter-PE networks. The sequencer consists of a sequence control program memory, program counter (PC), and branch control unit. The value of the program counter is identical to the address of context memories in the PEs.

In this architecture, a hierarchical structure of the cluster is adopted as a cluster group to process larger algorithms that do not fit within a cluster. Fig. 3.18 illustrates the cluster group. Inter-cluster networks are organized through crossbars. The sequencer in each cluster also controls the context switch of each crossbar which has five direction inputs/outputs (from/to upper, lower, left, right, and into/out of the cluster). There are no restrictions on the data transfer direction between the clusters, so the feedback paths that often appear in wireless communication baseband signal processing can be used. Each cluster can operate both independently and cooperatively in arbitrary combinations. Inter-cluster data transfer needs two or three additional cycles.

The shared resources located on the inter-cluster network are used by each cluster. There are two kinds of shared resources. One is a memory that can be shared by plural clusters. The other is dedicated hardware to accelerate signal processing.

The Fujitsu cluster architecture has been evaluated with algorithms that appear in the physical layer of IEEE802.11a and b wireless LAN processing. The evaluation results review that the cluster

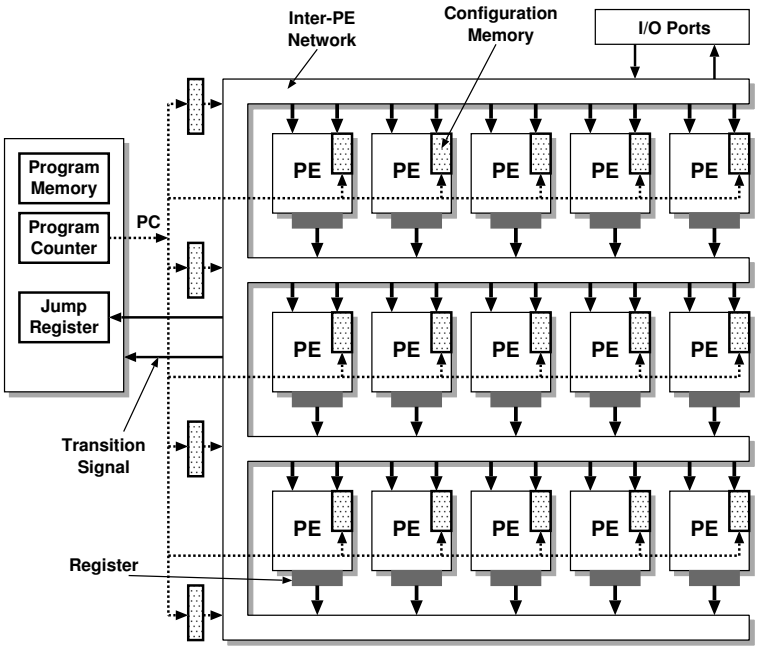


Fig. 3.17: Fujitsu Cluster Architecture

architecture provides a short latency compared to the DAPDNA-2 introduced in Section 3.3.3.

3.3.7 Elixent D-Fabrix

Elixent D-Fabrix[3] [70] reconfigurable processor is an evolution of the CHESS architecture [71] developed by Hewlett-Packard Laboratories.

The goal of the CHESS architecture was to increase both arithmetic computational density and the bandwidth and capacity of internal memories significantly beyond the capabilities of current FPGAs, while enhancing flexibility. Toward this goal, the fundamental computational unit is a 4-bit ALU with a primary set of 16 instructions. This provides efficient arithmetic capabilities suitable for cascading to useful media operations, or for supporting nibble-serial implementation styles. The entire user-visible routing structure is also based on 4-bit buses.

Fig. 3.19 shows the CHESS chessboard- or checkerboard-style reconfigurable arithmetic architecture. Each ALU is adjacent to four switchboxes, and each switchbox is adjacent to four ALUs. This allows very powerful local connectivity, with each ALU having input and output buses on all four sides, and being able to send data to or receive data from any of the eight surrounding ALUs as shown in Fig. 3.19 by the dotted arrows.

At run time, any switchbox in a CHESS array can be used as a 4-bit $\times$ 16-word memory. In this mode, all of the switches in the switchbox are disconnected, although the buses running over the switchbox can still be used. These memories are distributed through the array, attached to user plane wiring buses, and typically controlled by surrounding ALUs which generate an address and

[3]The technologies of the D-Fabrix are now licensed by Matsushita Electric Industrial Co., Ltd.

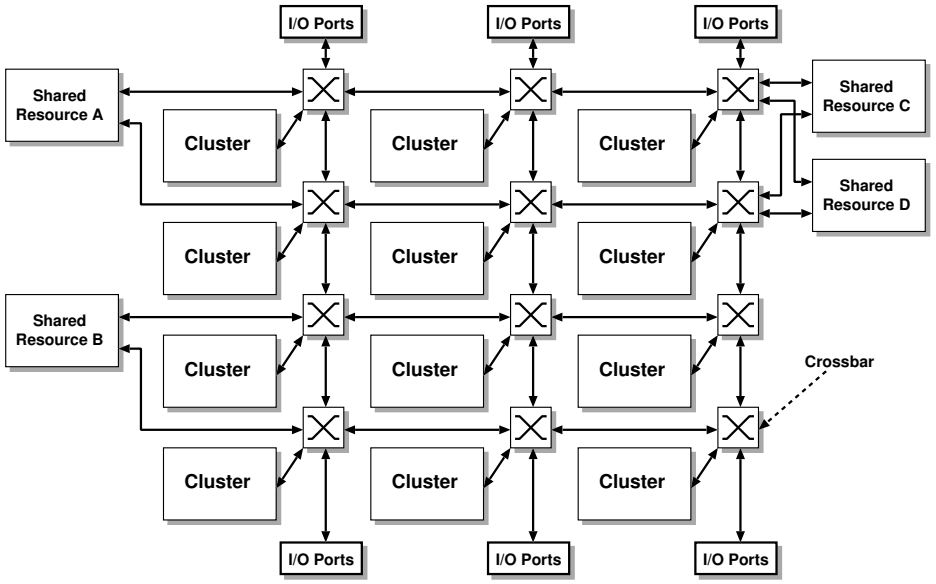


Fig. 3.18: Fujitsu Cluster Group

R/W control. The above design provides one block RAM per 16 ALUs[4], using about 25% more area than the basic array without the block RAMs.

Fig. 3.20 illustrates the ALU and Switchbox pair of Elixent D-Fabrix architecture. The architecture is almost the same as the CHESS architecture, for example, 4-bit ALU array and chessboard-style interconnections. There is, however, one significant difference between the D-Fabrix and the CHESS. The multiplexers that select signals from the routing network to pass to the pipeline registers are statically configured in the CHESS, but the D-Fabrix allows them to be dynamically controlled, using either an ALU Cout or a signal from the routing network as a multiplexer control signal.

The area of the basic cell (ALU plus switchbox) of the CHESS architecture is 0.045 mm^2 in a process with 0.35μm feature size. The simulation results show that an operational frequency is 200MHz, and the CHESS architecture has a substantial performance advantage over some FPGAs in an equivalent semiconductor technology. In the D-Fabrix architecture, dynamic switchbox multiplexers have resulted in a slight overall increase in the area of a combined ALU and Switchbox. This increase is due to the addition of the circuits that select control inputs to switchbox multiplexers and their configuration memories. The area increase is no more than 10%. Compared to the ALU, the switchbox multiplexer is one-third of the size and it has half the propagation delay and 40% of the power consumption.

The D-Fabrix ALU array is actually integrated into Toshiba's Media Embedded Processor (MeP) [72] and Matsushita's UniPhier platform [45, 46] as a flexible hardware engine. Toshiba developed a reconfigurable LSI called ET1, in which the D-Fabrix works as an extension unit of MeP Core which is a configurable processor core described in Section 2.3.2. Fig. 3.21 shows a block diagram

[4]Similar to MeP-combined version described later, in recent D-Fabrix arrays, the basic block called Tile consists of 2 ALUs and 2 Swichboxes, and every 2 x 4 Tile has one 256 bytes block RAM.

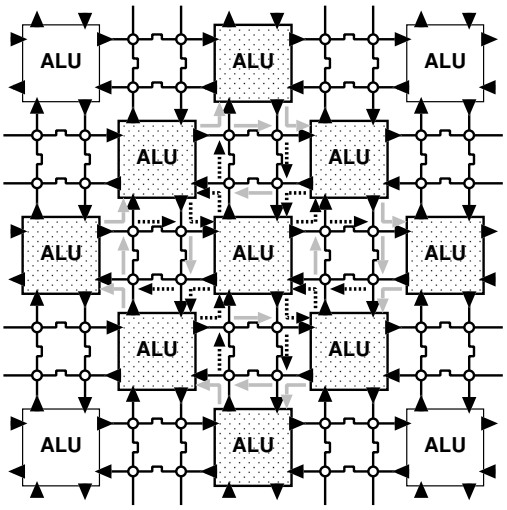


Fig. 3.19: Chessboard-style ALU Array

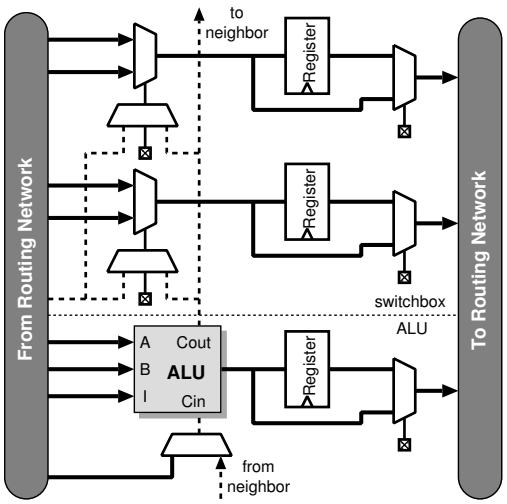


Fig. 3.20: D-Fabrix ALU and Switchbox

of ET1. The D-Fabrix of the ET1 consists of 576 Tiles, which means 1152 ALUs, and 72 Block RAMs. In [72], it is demonstrated that the area-efficiency of D-Fabrix compared to an ASIC ranges from 1/2.8 to 1/16.3, and the maximum operational frequency ranges from 1/1.7 to 1/5.8.

3.3.8 Rapport Kilocore

Rapport Kilocore [73] architecture is a commercialized reconfigurable processor derived from PipeRench [74], which has been proposed at Carnegie Mellon University in late 1990's.

PipeRench is based on a unique virtual hardware model called virtual pipelines described in Fig. 3.22. Fig. 3.22(a) shows a 5-stage (or stripe) virtual pipeline. Fig. 3.22(b) illustrates the first 7 cycles of reconfiguration of a 3-stage physical hardware pipeline executing the 5-stage virtual pipeline. Reconfiguration is performed by storing the configuration data of the entire virtual pipeline in an on-chip configuration data memory, and transferring them from the memory to the physical fabric every cycle.

Fig. 3.23 depicts a reconfigurable fabric of KC256, which is the first generation of the Kilocore architecture and virtual pipeline model. In Kilocore architecture, the functionality in each stripe consists of 16 8-bit PEs. All PEs within a stripe are interconnected within the stripe, which facilitates easier placement and routing of operations.

Fig. 3.24 is a block diagram of KC256's 8-bit PE. All selected inputs to multiplexers and shifters in this figure are connected to configuration bits stored in the PE. Special purpose interconnects are used to combine adjacent PEs to perform operations more than 8 bits in width. The shifters are also connected to the corresponding shifters in adjacent PEs to allow for efficient multi-PE shift operations. Each PE contains a register file with eight registers. The output from the functional unit can be written to any one register in the register file in the PE. If the value from the functional unit is not written to a register, the value from the corresponding register in the previous stripe is latched

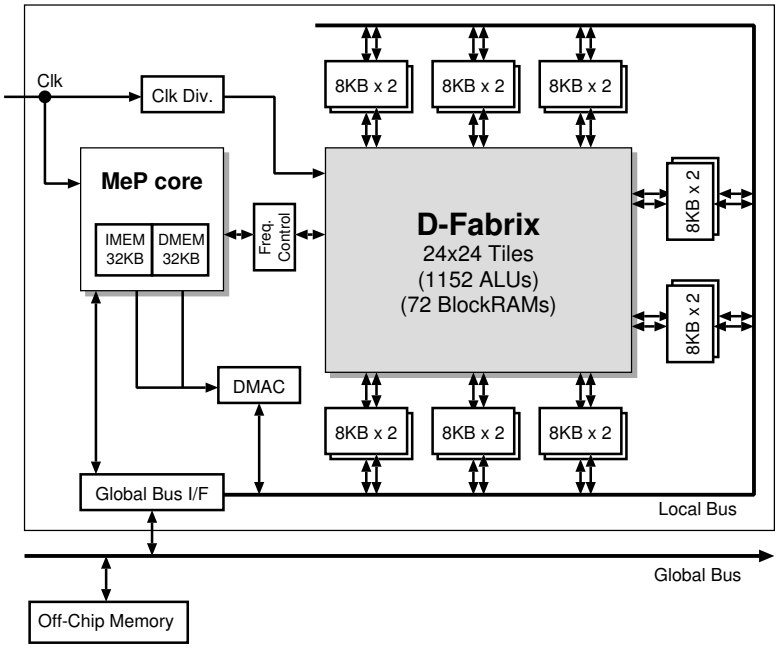


Fig. 3.21: ET1 Architecture with MeP core and D-Fabrix

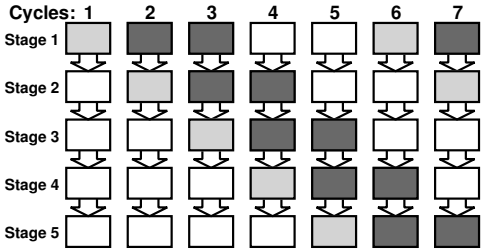
into the register. The functional unit in each PE consists of eight 3-input LUTs that are identically configured. The functionality of the PE is specified by 42 configuration bits, which means that each stripe has 672 bits of configuration.

The process of configuring a stripe takes one cycle, so the pipeline can be configured one cycle before the first data of the pipeline arrives at the stage. If the virtual pipeline is larger than the real hardware, physical stripes will eventually be reconfigured with new virtual stripes. The state of the over-written virtual stripes is preserved by writing the value in R0 into the R0 state store memory. The state will be restored when the escaped virtual stripe is returned to the fabric. The process of reconfiguration continues until the last stripe in the virtual pipeline is configured. After that, reconfiguration of a physical stripe starting with the first virtual stripe will occur, and the computation proceeds with new inputs.

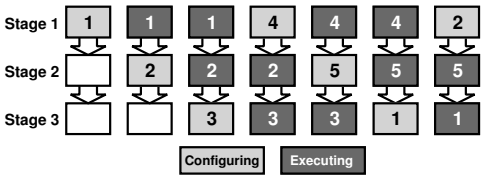
The KC256 is fabricated in a 6-metal layer 0.18 μm CMOS technology. The total die area is 7.3 $\times$ 7.6 mm^2, and transistor count is 3.65 million. The fabric clock is designed to operate at 120MHz under worst-case voltage and temperature conditions. At 120MHz, the KC256 executes a 40-tap 16-bit FIR filter at 41.8 mega samples per second, which means that its performance is in the same range as Texas Instruments C64x family DSPs. The KC256 performs the IDEA encryption and decryption at 450Mb/s. By comparison, an 800-MHz Pentium-III processor executes it at a rate of 75.4Mb/s.

3.3.9 PACT XPP-III

eXtreme Processing Platform (XPP) [75] is a reconfigurable processor architecture based on a hierarchical array of PEs called Processing Array Elements (PAEs). An XPP Core contains a rectangular



(a) 5-stripe Virtual Hardware Pipeline



(b) 4-Stripe Physical Hardware Pipeline

Fig. 3.22: Virtual Pipeline Model

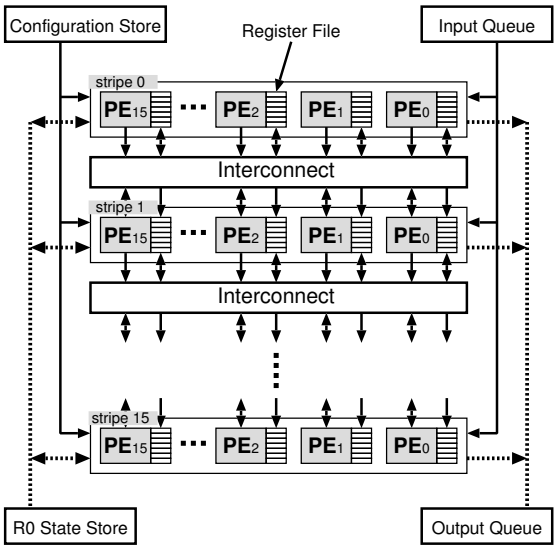


Fig. 3.23: Kilocore KC256 Architecture

array of three types of PAEs. Those in the center of the array are ALU-PAEs. To the left and right side of the ALU-PAEs are RAM-PAEs with I/Os. Finally, at the right side of the array, there is a column of FNC-PAEs. Fig. 3.25 shows a sample array with 30 ALU-PAEs, 12 RAM-PAEs, and 6 FNC-PAEs. The PAEs can be configured while neighboring PAEs are processing data. Reconfiguration is triggered by a controlling FNC-PAE or by event signals originating within the PE array.

The FNC-PAE comprises a 2×4 array of 16-bit ALUs, a Special Function Unit (SFU), a 16-bit register file, a 32-bit address generator (AG), a local instruction cache, a tightly coupled memory (TCM), and IO ports. The eight ALUs are designed to be small and fast because they are arranged in two non-pipelined columns of four ALUs each. A Special Function Unit (SFU) operates in parallel to the ALU datapath. It supports up to two 16x16-bit multiplications and functions such as a bit-field extraction. By combining the SFU multiplications with the adders of the ALU array, it is possible to execute two pipelined multiply-accumulate (MAC) operations each cycle.

As shown in Fig. 3.26, the ALU-PAE contains three XPP objects: FREG, ALU, and BREG object. All the objects have input registers which store the data or event packets for one cycle. The ALU object in the center of the PAE provides basic logical and arithmetic operations, and special arithmetic operations such as multiplication. The Forward Register (FREG) object on the left side and the Backward Register (BREG) object on the right side of the ALU-PAE are very similar. The main difference is the processing direction: top-down for the FREG and bottom-up for the BREG object. Both objects provide routing of data, dataflow operators such as multiplexing, basic arithmetic operations and look-up table for boolean operations.

The RAM-PAE consists of the FREG and BREG objects which are identical to the ones in the ALU-PAEs, a RAM object, an additional I/O object. The RAM object contains a small bank of two-ported SRAM. The RAM operates either in internal RAM (IRAM) or in a FIFO mode. The

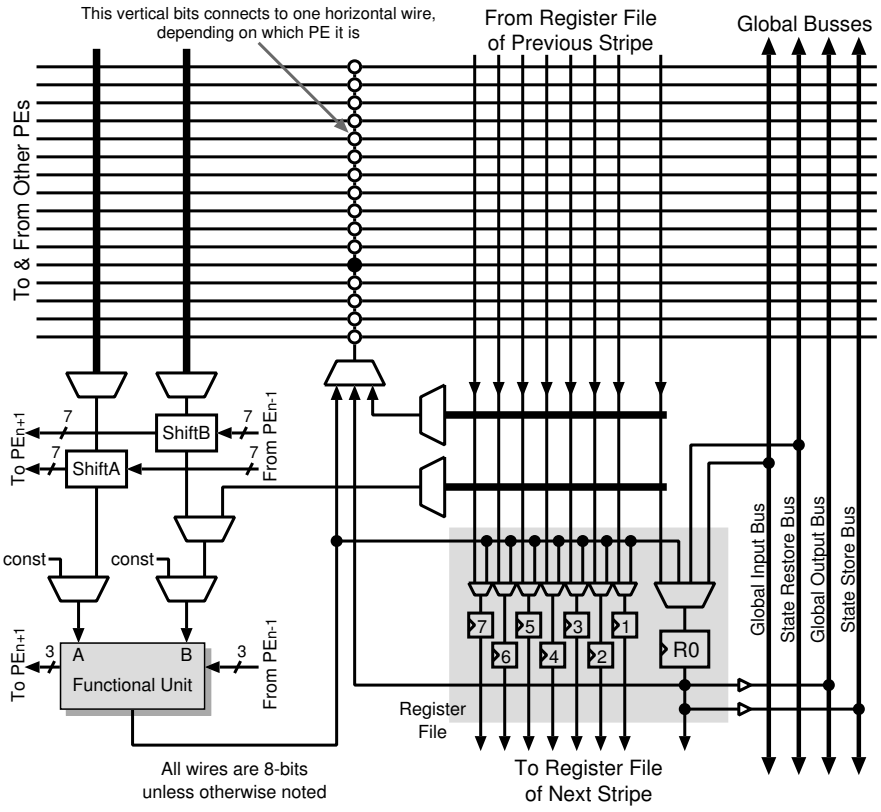


Fig. 3.24: Kilocore KC256 PE Architecture

content of RAMs is preserved during reconfiguration of the array. The I/O object is integrated into the RAM-PAE, providing access to external data.

The XPP objects communicate through a packet-oriented network. An operation is performed as soon as all necessary data input packets are available. The results are forwarded as soon as they are available, provided the previous results have been consumed. Thus it is possible to map a dataflow graph directly to ALU objects and to pipeline input data streams through it. The communication system is designed to transmit one packet per cycle. Hardware protocols ensure that no packets are lost, even in the case of pipeline stalls or during a configuration process.

In [76], it is described that a video decoder on the XPP-III, in which various video sequences including MPEG-2, MPEG-4, H.264, and VC-1 (WMV9), are supported. The evaluation result shows that the XPP-III version of 40 FNC-PAEs, 16 ALU-PAEs, and 8 RAM-PAEs can perform real-time decoding of H.264 frames with VGA size at 92MHz and HD resolution (1280x720) at 174MHz.

3.3.10 Stretch S5/S6 SCP Engine

The S6000 family configurable processors [77] are powered by Stretch S6 Software Configurable Processor (SCP) Engine. As shown in Fig. 3.27, they incorporate a Tensilica Xtensa LX dual issue

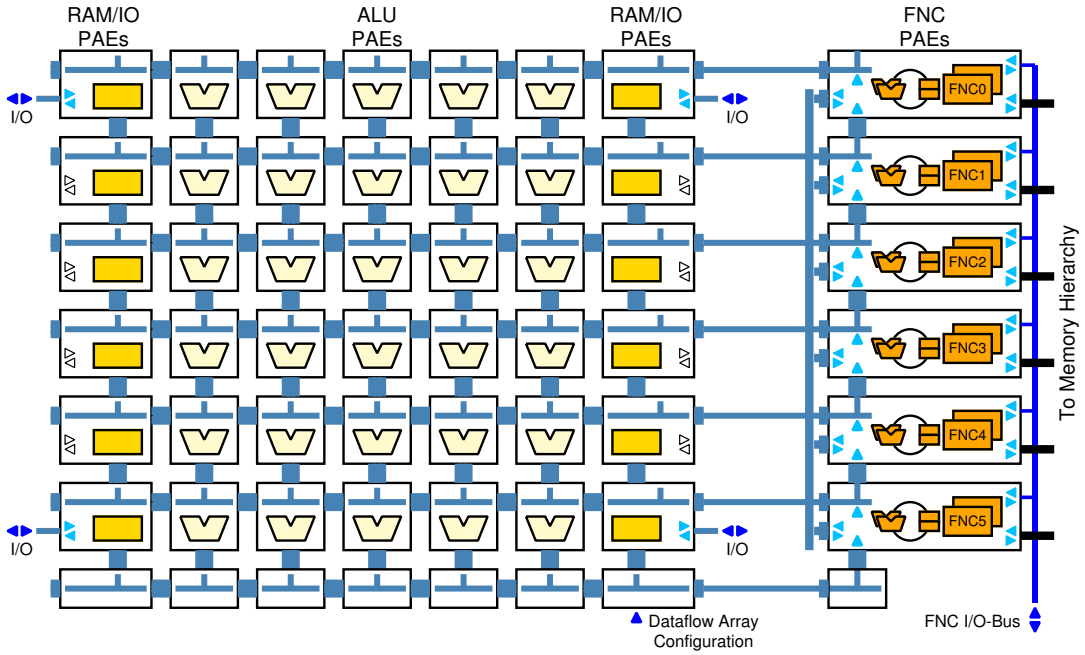


Fig. 3.25: XPP-III Core Architecture Sample

VLIW processor core and a second-generation Instruction Set Extension Fabric (ISEF). The ISEF is a software-configurable computing fabric that contains 64KB of embedded RAM (IRAM). Using the ISEF, system designers can extend the processor instruction set and define new instructions using only their C/C++ code. As a result, developers get the performance of custom hardware with C/C++ development simplicity.

The S6 SCP Engine within the S6000 family, described in Fig. 3.28, contains an Xtensa LX VLIW core and an ISEF. It is the ISEF that provides the dramatic application acceleration by allowing user algorithms to be instantiated in hardware and called by the processor as single instructions. The Stretch ISEF, being tightly coupled to the processor, only needs to host compute-based and logic functions.

The S6000 ISEF contains 4096 ALU-based PEs which, in addition to traditional ALU functions, can be configured to perform 2×4 multiplies and cascaded for larger data width. In addition, there are 64 dedicated multipliers capable of 8×16 operations that can be cascaded to increase data width. Distributed state registers provide local storage for intermediate values and coefficients. Connectivity of the PEs is enhanced with distributed multiplexers, priority encoders, and shifters.

For computation-intensive applications, the S6 ISEF is fed by the same 32 128-bit wide registers carried over from previous generations of Stretch devices [78]. These registers are used for loading data into the ISEF, and their presence in the S6000 ensures maximum compatibility and code reuse from previous software configurable processor designs. The S6000 ISEF also contains 64KB of embedded ISEF RAM (IRAM) distributed throughout the fabric in 32 banks of 2KB each. The IRAM is the memory mapped into the S6 SCP address space, so can be loaded directly by the

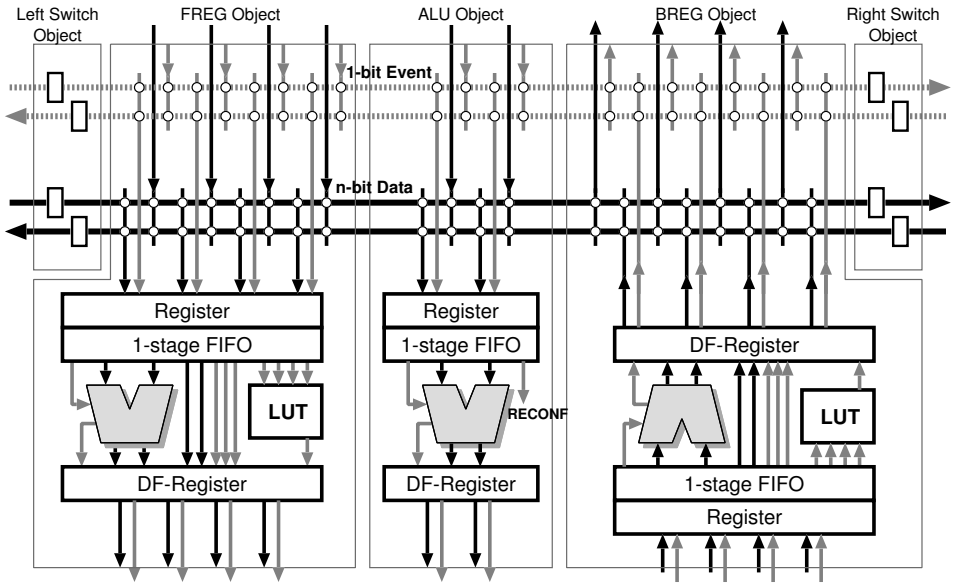


Fig. 3.26: XPP-III ALU-PAE Architecture

processor.

The ISEF supports dynamic reconfiguration based on the configuration delivery scheme. If a fetched opcode corresponds to an extension instruction that is not resident in the ISEF, an instruction fault is raised. The operating system will then save the contents of any internal state registers, find the extension instruction group containing the missing instruction, and initiate an ISEF reconfiguration before resuming the application program. The ISEF can be completely reconfigured in 27μs.

3.4 Summary

Table 3.1 summarizes the characteristics of the dynamically reconfigurable architectures described in this chapter. This table contains additional three academic architectures: PARS, MorphoSys, and WPPA. As shown in this table, a wide variety of architectures have been proposed, but they were designed for a particular purpose such as digital signal processing or wireless communication. The main reason is that the dynamically reconfigurable processors are expected to provide high area- and power-efficiency for the target application even with decreasing flexibility or programmability compared to FPGAs. Consequently, we can find various kinds of architectures, and the straightforward architectural classification has become complicated [7].

Dynamically reconfigurable processors have a very wide design space including PE structures, intra/inter-PE connection networks, number of available multipliers and distributed memory modules, dynamic reconfiguration methods, context memory capacity, PE-array sizes, and so on. Thus, unlike conventional FPGAs [1,2,21], it is difficult to explore a general-purpose and preferable architecture among the dynamically reconfigurable processors. Although they are just being in practical use, designers need to perform a time-consuming and inefficient architectural exploration for its tar-

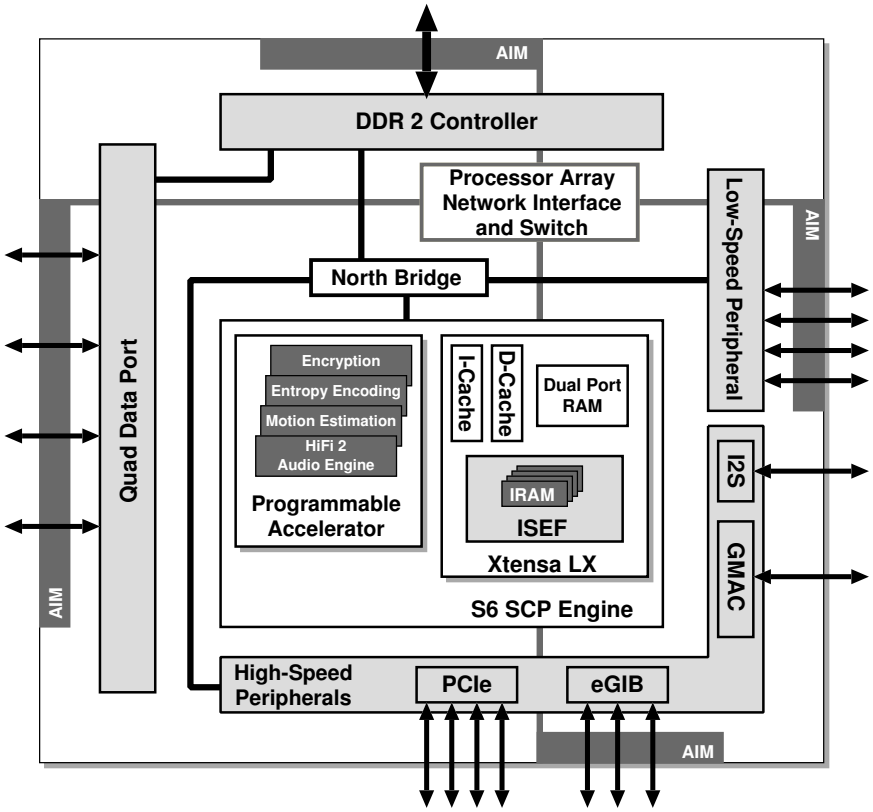


Fig. 3.27: S6000 Architecture

get application at design time.

As mentioned in Section 2.3, the configurable processors, in which several optimizations or customizations including instruction set extensions and memory size scaling can be performed, are fully exploited for embedded microprocessor designs. For also dynamically reconfigurable processors, a configurable architecture design methodology is needed in order to provide an efficient design space exploration to SoC designers. In WPPA architecture [57], each PE called weakly programmable processing element (WPPE) has a highly parameterized instruction set and a dynamically reconfigurable interconnection structure. With the help of configurable parameters, architectures optimized for a particular application domain can be designed. In addition, for efficient architecture exploration, analytical hardware cost model is derived in WPPA architecture design methodology.

This thesis aims to investigate speed and area/power trade-offs of the time-multiplexed execution models of the dynamically reconfigurable processors. By quantitatively investigating the hierarchical time-multiplexed execution models, we suggest a guideline to design an area- and power-efficient architecture for a particular application domain.

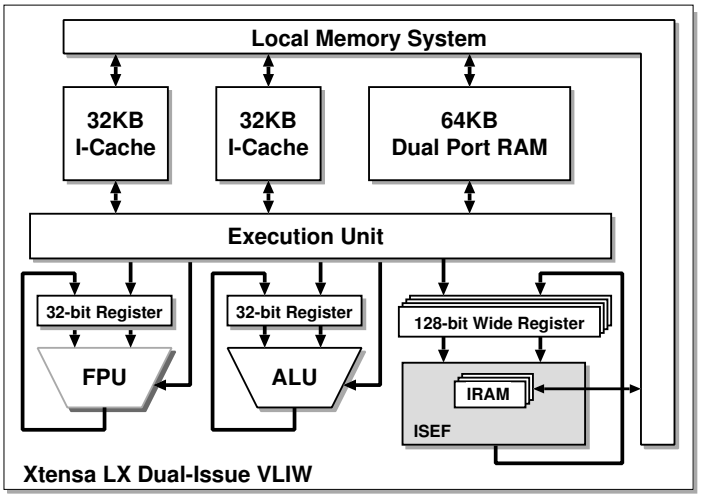


Fig. 3.28: S6 SCP Engine

Table 3.1: Summary of Recent Dynamically Reconfigurable Processors

Architecture	Vendor	Ref.	*G*	*S*	*R*	*I*
CS2112	Chameleon	[62]	16/32	108	Multi-Context (8)	Tiled, 2D Bus
DRP-1	NEC/NECEL	[63]	8	512	Multi-Context (16)	Tiled, 2D Bus
DAPDNA-2	IPFlex	[64]	32	376	Multi-Context (4)	Tiled, 2D Bus
DAPDNA-IMX	IPFlex	[66]	16	955	Multi-Context (4)	Tiled, 2D Bus
FE-GA	Hitachi	[67]	16	32	Multi-Context (4)	2D Direct
ADRES	IMEC	[68]	16	64	Multi-Context (32)	2D Direct
Cluster	Fujitsu	[69]	16	15/Cluster	Multi-Context (-)	3-Stage Switch
D-Fabrix	Elixent	[70]	4	576	Conf. Delivery	Chessboard
KC256	Rapport	[73]	8	256	Conf. Delivery	Xbar in Rows
XPP-III	PACT	[75]	16	Configurable	Conf. Delivery	2D Bus, Direct
S6 SCP Engine	Stretch	[77]	4/8	4096	Conf. Delivery	-
PARS UNITE	HCU[a]	[55]	8	20	Conf. Delivery	2D Bus
MorphoSys	UCI[b]	[56]	8/16	64	Multicontext (32)	2D Direct
WPPA	UEN[c]	[57]	Highly Parameterized and Configurable			

G = Granularity [bits],
S = PE-Array Size (# of PEs),
R = Dynamic Reconfiguration Scheme,
I = Inter-PE Connection Network

[a] Hiroshima City University
[b] University of California, Irvine
[c] University of Erlangen-Nuremberg

Chapter 4

Context-level Time-multiplexed Execution Model

This chapter explores a context-level time-multiplexed execution model which is based on NEC Electronics' DRP architecture introduced in Section 3.3.2. Firstly, Section 4.1 describes an overview of this research. The subsequent sections describe a basic model of the context-level time-multiplexed execution and an evaluation result of performance-area/power trade-offs.

4.1 Overview

The basic concept of time-multiplexed execution is based on the multi-context facility of reconfigurable devices. Recent multi-context dynamically reconfigurable processors can change contexts in a cycle-by-cycle manner as described in Section 3.1.2. The target application is divided into multiple contexts or tasks, and only the one required at that point can be activated and executed. The main objectives of time-multiplexing are as follows:

1. Executing large-scale programs which are difficult to be mapped onto a single reconfigurable fabric, or supporting various kinds of programs at run-time.

2. Saving physical circuit area and power consumption while retaining required performance by time-multiplexing a target program,

The basic concept of the time-multiplexed execution was introduced in early 1990's, and it is based on fine-grained reconfigurable devices such as FPGAs [49,51]. Since the proposal of WASMII in 1992, we have investigated the dynamically reconfigurable architecture aimed to satisfy Objective 2, but many researches have revealed that fine-grained FPGA-based time-multiplexed execution is not efficient in area and power consumption [79]. Hence, it could not be actually applied to cost- and power-intensive embedded systems. Consequently, the fine-grained time-multiplexed reconfigurable systems target logic emulation or prototyping of large-scale applications such as scientific computations and system simulations, and Objective 1 is appropriate for these systems.

In the last decade, a widespread use of multimedia applications and mobile terminals widened the application field of coarse-grained dynamically reconfigurable processors, which are mainly focused

on this thesis. They are expected to be used for cost- and power-intensive embedded systems such as cell phones and portable game machines. Thus, the main objective of the time-multiplexed execution on the dynamically reconfigurable processors is Objective 2.

In addition, they have some additional differences from time-multiplexed FPGAs. An equivalent circuit corresponding to a single context of commonly used dynamically reconfigurable processors is relatively small compared to FPGAs. For reconfiguration time, while the FPGAs require millisecond-order latency, dynamically reconfigurable processors demand smaller microsecond- or nanosecond-order latency. Hence, it is significant that an expected circuit scale and an interval of the dynamic reconfiguration are different between FPGAs and dynamically reconfigurable processors.

This chapter describes the context-level time-multiplexed execution model which aims to satisfy Objective 2. A dynamically reconfigurable processor core is expected to be installed into embedded SoCs. The dynamically reconfigurable processor itself is user-programmable for target applications after the SoC fabrication. However, SoC designers must define a PE-array size and the number of contexts at the design time so that the area- and power-efficiency are optimized for the target applications. These two architectural parameters are quite significant because they directly settle the performance, area, and power consumption on the context-level time-multiplexed execution. Therefore, the PE-array configuration which achieves high area- and power-efficiency for the target applications should be clarified at the SoC design time.

For common processors, as described in Section 2.3, the technologies of configurable processors are widely used. However, there is little literature which demonstrates quantitative evaluations with respect to area- and power-efficiency of the context-level time-multiplexed execution scaling the PE-array size. So, the design methodology for the optimal PE array has not been established well.

In this work, the trade-offs between performance and area/power of the context-level time-multiplexed execution is analyzed by scaling the PE-array size. The target architecture is NEC Electronics' Dynamically Reconfigurable Processor (DRP) which was described in Section 3.3.2. We make a performance and cost model of the context-level time-multiplexed execution. And also, we propose an analytical method to estimate the optimal size of PE array for area- and power-efficiency satisfying a performance requirement of a target application. In this estimation method, an application parallelism is firstly evaluated assuming infinite number of PEs. And then, performance and cost are estimated by scheduling contexts for each PE-array size. The DRP-1 chip and the DRP compiler are used, and various benchmark applications are implemented scaling the PE-array size.

4.2 Basic Model

4.2.1 Parallelism Diagram

The target application field of DRP contains streaming and multimedia applications for embedded systems. In multi-context devices, a target application is divided into multiple contexts, and they are

executed in sequential manner at run-time. Although context switching is performed sequentially, every PE and distributed memory within a context can operate in parallel. Hence, the performance of the target application actually depends on the context scheduling.

In general, computing algorithms have inherent parallelism and sequentiality, and so certain clock cycles are required for executing the algorithm. In particular, when the DRP core executes a stream-based application, there exists the following possible processing flow:

1. Data are read out from distributed memory modules and/or registers,
2. required processing is executed on the PE array, and then
3. the results are written back to distributed memory modules and/or registers.

Here, these contiguous processes are considered as a step in the algorithm. Usually each step is iteratively executed and requires at least a clock cycle since the results must be written into distributed memory modules and/or registers. After finishing all steps, the output data stream is flushed out from the chip. In the context-level time-multiplexed execution model, the steps are mapped onto contexts of the PE array, and step transitions correspond to the context switching.

The target application has a restriction of the sequential execution, and parallelism is defined for every sequentially executed step. Throughout this work, we consider the number of required PEs as the parallelism for each step. Fig. 4.1 shows a sample program of Discrete Cosine Transform (DCT) in JPEG encoder, and Fig. 4.2 draws the distribution of the number of required PEs in each sequential step when the program is mapped onto the DRP core. For each application, we refer to the distribution of parallelism as *parallelism diagram*. In the parallelism diagram, PEs and VMEMs/HMEMs are assumed to be available without any restrictions. Note that, each step may be iteratively executed several times according to the algorithm.

The DCT algorithm mainly performs product-sum operations in row and column directions on an 8×8 image data. The behaviors of each step are as follows.

- Initialize distributed memory modules and registers, and input data is stored (Step 0-3).
- Operate in the row direction (8-times loop of Step 4-7).
- Operate in the column direction (4-times loop of Step 8-15 and then Step 16-23).
- Output results (Step 23-25).

Note that the program shown in Fig. 4.1 corresponds to the row-directional operations (Step 4-7).

In this implementation, an 8×8 image data is stored into eight independent VMEM-based arrays so that they can be accessed in the row direction simultaneously. Once the data is ready, row-directional operations can be executed fully using nearly 600 PEs. In the DCT design, as shown in Fig. 4.1, the multiplier factor is constant, and in such a case, the multiplication is transformed into shifts and additions automatically by the DRP compiler. Although the column direction access

```
// for 8 x 8 pixels
for(i = 0; i < 8; i++) {
  SUM0 = (data[i*8+0] + data[i*8+7]);
  SUM1 = (data[i*8+1] + data[i*8+6]);
  SUM2 = (data[i*8+2] + data[i*8+5]);
  SUM3 = (data[i*8+3] + data[i*8+4]);

  SUB0 = (data[i*8+0] - data[i*8+7]);
  SUB1 = (data[i*8+1] - data[i*8+6]);
  SUB2 = (data[i*8+2] - data[i*8+5]);
  SUB3 = (data[i*8+3] - data[i*8+4]);

  // constant multiplications
  Z0 = A*SUM0 + A*SUM1 + A*SUM2 + A*SUM3;
  Z2 = B*SUM0 + C*SUM1 - C*SUM2 - B*SUM3;
  Z4 = A*SUM0 - A*SUM1 - A*SUM2 + A*SUM3;
  Z6 = C*SUM0 - B*SUM1 + B*SUM2 - C*SUM3;

  Z1 = D*SUB0 + E*SUB1 + F*SUB2 + G*SUB3;
  Z3 = E*SUB0 - G*SUB1 - D*SUB2 - F*SUB3;
  Z5 = F*SUB0 - D*SUB1 + G*SUB2 + E*SUB3;
  Z7 = G*SUB0 - F*SUB1 + E*SUB2 - D*SUB3;

  data[i*8+0] = Z0 >> 14;
  data[i*8+1] = Z1 >> 14;
  data[i*8+2] = Z2 >> 14;
  data[i*8+3] = Z3 >> 14;
  data[i*8+4] = Z4 >> 14;
  data[i*8+5] = Z5 >> 14;
  data[i*8+6] = Z6 >> 14;
  data[i*8+7] = Z7 >> 14;
}
```

Fig. 4.1: DCT Algorithm (row direction)

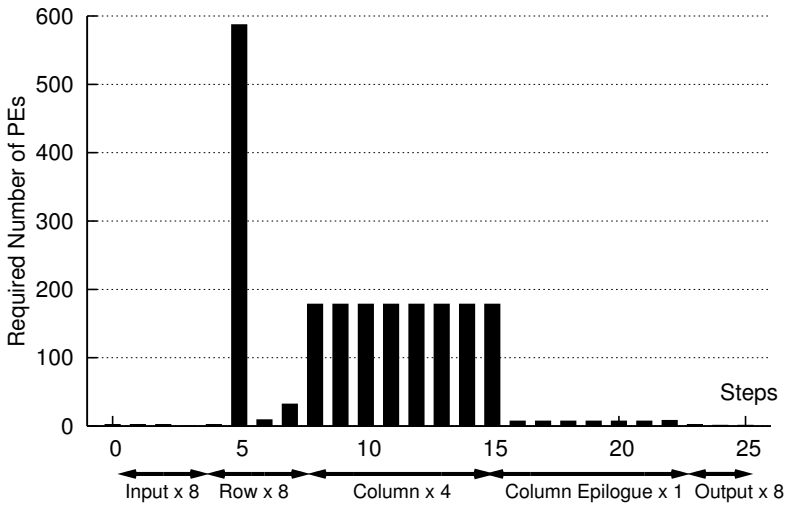


Fig. 4.2: Parallelism Diagram of DCT Design ($N = \infty$)

must be performed in a sequential manner, the pipelined accessing of multiple arrays can enhance the parallelism. Therefore, the sequential memory accesses in a column direction do not degrade the parallelism so much except for the last iteration to store final results.

4.2.2 Performance and Cost Modeling

In the context-level time-multiplexed execution, it is a straightforward interpretation that a step operation is mapped to a single context of the PE array and a step transition corresponds to its context switching. Here, the performance and cost are modeled based on the parallelism diagram.

At first, we define a context size N as the number of available PEs for a context. Then, let S_N be a set of steps of an application. For all step $i \in S_N$, the number of required PEs, iteration count, and delay time are given by n_N^i, l_N^i, t_N^i, respectively. We can interpret n_N^i as the parallelism of step $i \in S_N$. Let $n_N^{\max}$ be the maximum number of required PEs of all steps for each N, and we have

$$n_N^{\max} = \max_{i \in S_N} \{n_N^i\}.$$

If we assume that infinite number of PEs are available (i.e., $N = \infty$), similar to the example shown in Fig. 4.2, $n_\infty^{\max}$ means the maximum parallelism which the target application originally has. As mentioned below, the context size N must be selected so that $n_N^{\max} \leq N$[1].

If the execution clock cycles and maximum delay time are denoted by c_N and d_N, respectively, we have

$$c_N = \sum_{i \in S_N} l_N^i, \qquad d_N = \max_{i \in S_N} \{t_N^i\}.$$

The execution time of the application e_N can be calculated by the product of c_N and d_N. If $N = \infty$, c_∞ means minimum number of required steps considering iterations for executing a certain application

[1] In the actual design, the context size N should be chosen so that approximately $n_N^{\max} \leq 0.8N$ for sufficient routability.

with infinite PEs and VMEMs/HMEMs. In the case of the DCT design, for example, we have $c_\infty = 72$ ($4 \times 8 = 32$ for each direction, 8 for an epilogue of the column direction, and except for input/output). Since c_∞ depends on the algorithm and the data structure, the DCT design requires 72 clocks even with the infinite number of available PEs and VMEMs/HMEMs. Therefore, c_∞ also means the minimum number of execution clock cycles required for the target application.

Here, we can consider the case that all steps are realized on a single context assuming that available PEs and VMEMs/HMEMs are boundless. This corresponds to the case without the time-multiplexed execution. Even in this case, the target application still requires c_∞ clock cycles, and then the execution time is represented as $e_\infty = c_\infty d_\infty$.

In the context-level time-multiplexed execution, a step operation is mapped to a context of the PE array, and a step transition corresponds to its context switching. However, since the number of available PEs in a context and the number of contexts are actually finite, context scheduling techniques are actually needed to improve the resource utilization.

4.3 Context Scheduling Techniques

As mentioned above, in the context-level time-multiplexed execution model, each step of a target application is assigned to a context. However, as shown in Fig. 4.2, a target application has nonuniform parallelism diagram for each step. If a context size is decided so as to match the context with maximum PE usage, it is very inefficient with respect to hardware area and power consumption. Hence, step-context allocations are actually optimized by using the following context scheduling techniques.

4.3.1 Step Division

For each application, the parallelism diagram is defined as shown in Fig. 4.3(a). In the time-multiplexed execution, each step is assigned to a context under the constraint of the number of available PEs per context, i.e., context size N. This constraint can be represented by $n_N^i \leq N$, where n_N^i is parallelism of step i. If the number of required PEs for some step exceeds this constraint, the step must be subdivided into child steps as shown in Fig. 4.3(b). We call this technique a *step division*.

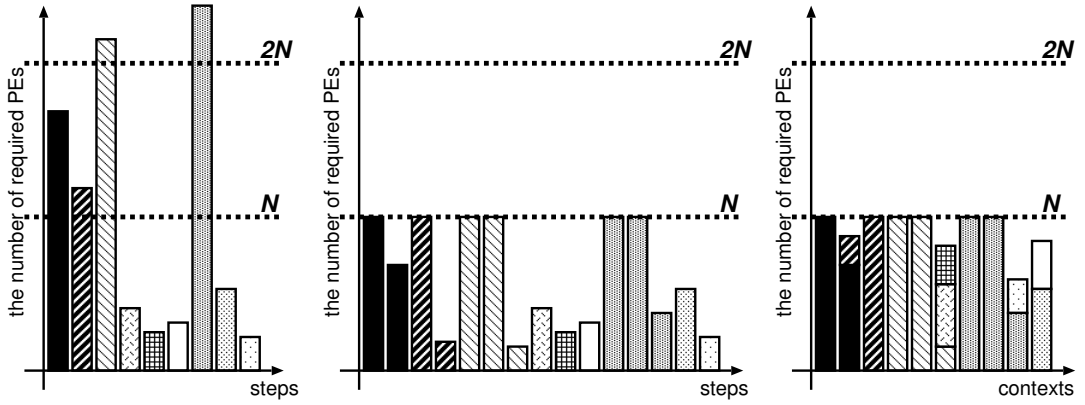
Given N, by the step division, the minimum number of required steps s_N is defined by

$$s_N = \sum_{i \in S_\infty} \left\lceil \frac{n_\infty^i}{N} \right\rceil \geq s_\infty.$$

And also, for the execution clock cycles c_N, we have

$$c_N = \sum_{i \in S_N} l_N^i = \sum_{j \in S_\infty} \left(\left\lceil \frac{n_\infty^j}{N} \right\rceil l_\infty^j \right) \geq c_\infty.$$

The above model can estimate execution clock cycles of the target application when a context size N is specified. For delay time, if the step division cuts a maximum delay path for each step and



(a) Original Parallelism (b) Step Division (c) Multiple-Step Allocation

Fig. 4.3: Context Scheduling Techniques

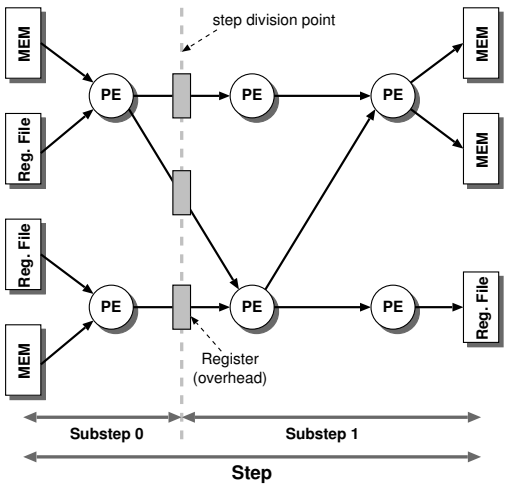


Fig. 4.4: Cost Overhead of Step Division

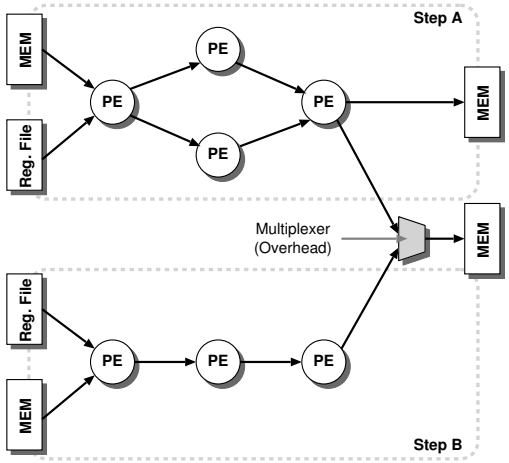


Fig. 4.5: Cost Overhead of Multiple-Step Alloc.

reduces the delay time at a constant rate, the execution time e_N can be also estimated by this model. In general, however, the effects of the step division to the delay time for each step are all different.

In addition, as overhead of the step division, additional registers are inserted at division points for inter-step communications as shown in Fig. 4.4. Furthermore, in the case that more divisions are not feasible, like constant multiplications, inefficient division results in increasing resource usage and enlarging the delay time. This trade-off is discussed based on an evaluation of real benchmark applications (see Section 4.5).

4.3.2 Multiple-Step Allocation

As shown in Fig. 4.2, there exist many steps which use few PEs because they handle memory access only. The PE-array utilization is quite poor in this DCT design. Furthermore, the number of required contexts will also exceed the capacity of the DRP-1 (i.e. 16 contexts). To overcome this problem, a

multiple-step allocation method packs multiple steps which have many unused PEs into one context keeping the context size fixed. Fig. 4.3(c) shows the concept of the multiple-step allocation method.

A STC of the DRP core determines the step to be activated according to a state transition table, and the other steps in the context are disabled. If all of the steps for a target application are assigned to a single context, this corresponds to the normal execution model without time-multiplexing. But, in the case of DRP, the number of integrated steps to a context is restricted to a maximum of 4. So, this technique can improve the PE-array utilization and reduce the number of contexts. Although it is difficult to formulate the concept of the multiple-step allocation, the number of required contexts x_N can be estimated by a simple program[2] because this method just determines the integrated steps under a certain context size N.

This technique surely contributes to improve PE-array utilization and reduce the number of required contexts, but never improve the execution time. The main reason is that s_N and c_N remain unchanged even if the multiple-step allocation is fully exploited. In addition, since a whole datapath on a single context is spatially extended, the wiring delay usually grows longer. Furthermore, additional multiplexers are necessary to share registers and distributed memory modules among the integrated steps as shown in Fig. 4.5. This results in the increase of required PEs per context and the maximum delay time. Consequently, the execution time may be damaged by the multiple-step allocation.

For this reason, unnecessary multiple-step allocations should be avoided. Fortunately, since the step including the maximum delay path commonly requires many PEs by itself, it is never selected as a target step of the multiple-step allocation. Therefore, it has little impact on the delay time as far as abundant multiple-step allocations are not performed. In order to analytically estimate the maximum delay time among all steps, we assume that the multiple-step allocation does not stretch the maximum delay path ideally.

4.4 Context Size Scaling and Trade-offs

The context-level time-multiplexed execution based on the multi-context functionality provides high area- and power-efficiency to embedded SoCs. This is derived from the property that a required context can be activated only when necessary and unrequested contexts are temporary hidden behind the required context. Thus, a target application can be executed consuming minimum hardware cost and power for the required performance. The context size is a crucial factor for the time-multiplexed execution to affect the performance, area, and power consumption. It is important to determine the optimal context size for the target application in order to optimize area- and power-efficiency.

If a context size is small, the physical hardware area is also small, but the step division results

[2]The calculation of the number of required contexts with the multiple-step allocation is originally an NP-hard problem, *bin-packing problem*. Fortunately, since the number of steps which can be integrated into one context is small enough to be solved with realistic complexity.

in increase of the required number of steps and contexts. Meanwhile, the operational frequency can be improved because the maximum delay path is also divided by the step division. If a context size is small, the power consumption of running PEs can be also small. However, improvement in operational frequency can increase clock network power, and frequent context switching triggers the increase of dynamic power consumption.

On the other hand, a large size of context helps more parallel processings with more PEs and VMEMs/HMEMs. This leads to achieve high throughput and reduce the required steps and contexts. Although low operational frequency and low-frequent context switching can reduce power consumption, running PEs including ones in which any operations are not assigned increase the power consumption. Given an excessive context size, inappropriate multiple-step allocation may cause fatal overhead of delay and resource cost because of increasing multiplexers as mentioned in Section 4.3.2.

When we select the context size for a target application, we must consider the above architectural trade-offs for area- and power-efficiency. The following section describes quantitative evaluation results of the trade-offs based on real application designs on the DRP-1.

4.5 Application Implementations and Results

This section describes quantitative evaluation results of real application implementations and comparison to the estimation model.

4.5.1 Benchmark Applications

The following applications are targeted in this work:

- Discrete Cosine Transform in JPEG encoder (DCT) [80]
- Inverse Modified Discrete Cosine Transform in MP3 decoder (IMDCT) [80]
- Fast Fourier Transform (FFT) [81, 82]
- Viterbi decoder (Viterbi) [14]
- Advanced Encryption Standard (AES) [12, 13]
- Secure Hash Algorithm 1 (SHA-1) [12, 13]

The above applications have been described in BDL, which was introduced in Section 3.3.2, and they are compiled by the DRP Compiler.

In order to evaluate the trade-offs of the context-level time-multiplexed execution model, each application is designed scaling a context size N. In the case of the DRP architecture, the context size can be expanded by the Tile. Specifically, at first, parallelism of each application is measured under $N = \infty$. And then, under $N = 512$ (i.e., 8 Tiles of DRP), each application is designed optimizing

data structures to increase the parallelism within the context size. Next, with decreasing the context size N, each application is optimized for each N.

The context scheduling schemes including the step division and multiple-step allocation can be performed automatically or manually. In this experiment, we optimize the step division for each context size manually so that the performance is maximized. In contrast, the multiple-step allocation, as mentioned above, has no significant negative effects to performance and cost unless abundant allocations will not performed. In the case of DRP, the DRP compiler can perform the multiple-step allocation without increasing cost, and the preferable result can be obtained by an automatic allocation with the DRP compiler compared to a manual allocation. Hence, we leave the multiple-step allocation to the DRP compiler unless the application cannot be designed within 16 contexts or it causes crucial enlargement of maximum delay time.

In addition, for some $i \in S_\infty$, if the number of required PEs exceeds 8 Tiles, i.e., $n_\infty^i > 512$, the place-and-route phase for the application cannot be performed. In this case, we use the results from pre-place (post technology mapping) phase instead. Compared to the result after place-and-route, while step and context counts are not different, the larger number of required PEs tends to be calculated because a circuit optimization is performed at the place-and-route phase.

Based on a parallelism diagram with $N = \infty$ and the context-level time-multiplexed execution model shown in previous sections, we estimate PE and context usage and execution time, and then compare to implementation results for each context size N.

4.5.2 Results and Comparison to the Model

Table 4.1 summarizes the implementation results including context size (#Tile), maximum number of consumed PEs of all contexts ($m_N^{\max}$), maximum delay time, required execution clock cycles, and power consumption for each application. Note that values in parentheses show estimation results from the context-level time-execution model. For #Tile = ∞, the evaluation results under unbounded PE array without context scheduling are shown.

Note that $m_N^{\max}$ is different from the maximum parallelism $n_N^{\max}$ for each N. While $m_N^{\max}$ represents the maximum number of required PEs of all contexts, each context can contain several steps. However, we must satisfy the context size constraint for $m_N^{\max}$, i.e., $n_N^{\max} \leq N$. And, if $N = \infty$, $m_N^{\max}$ become completely identical to $n_N^{\max}$.

As described above, $n_\infty^{\max}$ means the maximum parallelism of the target application. From Table 4.1, we can claim that DCT, IMDCT, Viterbi, and AES-ECB have relatively larger parallelism than FFT and SHA-1. The maximum delay time d_N was obtained from a report after place-and-route with the DRP compiler. The number of clock cycles c_N was calculated as execution clock cycles required for processing one block data of input stream. The power consumption p_N was estimated by a power profiler of the DRP compiler. The power profiler can estimate the power consumption based on simulation and compiled results such as PE usage and maximum operational frequency.

Table 4.1: Implementation Results of Benchmark Applications

Application	#Tile ($N/64$)	Context (x_N)	Max PEs $\left(m_N^{\max}\right)$	Delay[ns] (d_N)	Clocks (c_N)	Power[mW] (p_N)
DCT	∞	26	588	87.7	73	-
	8	16 (10)	259 (379)	82.0	92 (81)	345.4
	6	25 (14)	186 (304)	76.0	153 (81)	269.0
	4	34 (16)	148 (200)	67.3	168 (89)	217.6
	2	64 (27)	73 (98)	69.2	332 (145)	113.1
IMDCT	∞	16	888	160.6	1347	-
	8	13 (9)	360 (333)	95.5	3681 (3773)	248.3
	6	14 (11)	280 (301)	102.4	3791 (3837)	180.1
	4	20 (14)	183 (198)	83.7	3928 (3901)	164.7
Viterbi	∞	3	893	60.0	2	-
	8	7 (4)	320 (364)	27.2	8 (4)	1136.0
	6	8 (5)	204 (298)	27.7	10 (4)	908.1
	4	9 (7)	160 (179)	31.0	11 (6)	611.8
AES-ECB	∞	8	514	29.1	10	-
	8	6 (4)	448 (357)	27.0	20 (20)	1092.5
	6	7 (6)	224 (257)	22.2	29 (20)	1080.0
	4	8 (7)	224 (172)	27.6	29 (30)	627.3
FFT	∞	14	61	40.1	49050	-
	8	5 (4)	101 (123)	35.6	49050 (49563)	731.4
	4	6 (4)	87 (123)	37.2	49050 (49563)	373.2
	2	7 (5)	73 (83)	38.4	49050 (49563)	225.3
	1	16 (7)	33 (51)	23.0	73125 (60827)	186.9
SHA-1	∞	14	61	35.1	93	-
	8	8 (4)	61 (217)	27.6	93 (93)	806.8
	6	8 (4)	61 (217)	27.2	93 (93)	624.1
	4	8 (4)	61 (156)	29.5	93 (93)	394.8
	2	8 (5)	61 (94)	29.6	93 (93)	213.8
	1	10 (10)	53 (46)	31.2	94 (135)	114.3

But, the estimation can be different from the actual measurement using a real DRP chip.

From the evaluation results, as the context size N decreases, the required contexts x_N increase and $m_N^{\max}$ decreases. We can see a similar trend in the analytical estimation method shown in the previous section. However, since the estimation method does not consider overhead caused by the step division and the multiple-step allocation, there exist some differences between the estimation and the actual measurement. In addition, since the model ignores increase of delay time caused by the multiple-step allocation, small-scale applications such as FFT and SHA-1 are modeled so that the required contexts are reduced by multiple-step allocation as much as possible. In the actual implementations, however, each design is optimized for the target context size N. Thus, multiple-step allocation is not performed if it negatively affects delay time and performance. Therefore, we can see that $m_N^{\max}$ is smaller and required contexts x_N is larger than the estimation results.

On the other hand, the execution clock cycles, similar to the estimation result, are increases with increasing context size N. But, in the case of highly-parallel applications such as DCT and Viterbi, there exists a big difference from the estimation results. We can guess that the overhead of the step divisions becomes critical because unnecessary step divisions insert many registers.

In the estimation model, we assumed that the maximum delay time decreases at a certain rate due to step division and also multiple-step allocation has no negative effects on delay time. However, their effects on the delay time completely depend on the characteristics of the target application such as communications patterns between PEs. Although both FFT and SHA-1 have a small degree of parallelism, in the case of FFT, required contexts can be reduced with larger context sizes because multiple-step allocation gives few critical overhead to the delay time. In contrast, since the multiple-step allocation would enlarge the maximum delay time of SHA-1 and it would not be improved even with larger context sizes, and consequently the required contexts are not reduced. In general, the maximum delay time can be reduced by step divisions, but we can see that the maximum delay times of Viterbi, SHA-1, and FFT more increases. This is because of increasing routing wires caused by increasing resource density and descending flexibility of place-and-route with decreasing context size N. In addition, we can see unexpected slight increases because the delay time completely depends on mapping and routing algorithms of the DRP compiler.

In the model, it is difficult to predict the power consumption. From the evaluation results shown in Table 4.1, we can see the trend that the power consumption also decreases with decreasing context size even in small-scale applications. The most plausible reason is that the total amount of operated PEs, including ones which have no instruction to execute, increases with larger context sizes.

By using the DRP simulator and the power profiler, we have carried out the following simplified experiment to estimate the power consumption of the context switching. At first, the following two designs are implemented on the DRP-1 device.

- Design (A) consists of an infinite iteration of only an empty context in which all PEs have no instructions, and

- Design (B) consists of an infinite iteration of two equivalent empty contexts, and each of the contexts switches to the other cycle-by-cycle.

For these designs, operational frequency and execution clock cycles are fixed, and then the power consumptions are compared. As a result, the increase of the power consumption of Design (B) over Design (A) was only 5%. This result suggests that the power consumption of fundamental parts such as PE operations and a clock network are dominant, and the frequency of context switching is not so influential for the total power consumption. Therefore, it is disadvantageous in terms of the power consumption with the increasing context size. Note that additional resources such as costs of the step division and the multiple-step allocation cause an increase of power consumption.

4.6 Trade-off Evaluation

This section describes performance and area/power trade-offs for each context size and benchmark applications. At first, evaluation metrics are defined in the next subsection, and then evaluation results are shown.

4.6.1 Evaluation Metrics

Firstly, the performance of each application for a given N is measured by the execution time $e_N = c_N d_N$.

Next, the measurement of area cost is much more complicated. The area cost for each application is defined by the total area of used Tiles. It involves the area of PEs and VMEMs/HMEMs used in the application and the context memory of each PE. In the case of the DRP architecture, a Tile which is composed of 64 PEs equips the constant number of VMEMs and HMEMs. Their costs are linearly increased for the number of Tiles, i.e., context size N. For a given context size N, the area cost f_N is defined by

$$f_N = (1 + \gamma x_N) \left\lceil \frac{N}{64} \right\rceil,$$

where γ means the ratio of the context memory area per context to the datapath area of a PE.

Although unfortunately the area ratio of the DRP-1 device is undisclosed, in the case of 16-bit dynamically reconfigurable processor ADRES [68] from IMEC, the area of a context memory for 32 contexts is almost equivalent of the area of a PE. In this case, $\gamma = 0.031$. While the data width of the ADRES is 16-bit, in contrast, the DRP has 8-bit PEs. In general, datapath area is proportional to data width but the increase of configuration data is not so large compared to the increase of the datapath area because the wide data width results in an increase in instruction width of ALU and DMU. Therefore, γ becomes small with the increasing data width of PE. Unfortunately, we cannot find the official configuration data format of DRP and ADRES, we assumed that γ for the DRP

architecture can be estimated to be approximately more than twice larger than that of the 16-bit ADRES architecture, and simply decided to be $\gamma = 0.1$.

In this evaluation, furthermore, we assume that the number of required contexts x_N is smaller than that of available contexts in the target DRP core. Although the current DRP-1 has 16-context instruction memory, next-generation architectures can be assumed to support about 64 contexts without critical architectural changes. Hence, we assume $x_N \leq 64$ throughout this work.

Assuming the above evaluation settings, we define the required area cost for the performance of the context-level time-multiplexed execution, r_N, as $r_N = f_N e_N$.

Finally, power-efficiency is measured by required energy consumption for executing each target application because battery loads should be cared for embedded mobile devices. We define the required energy consumption j_N as the product of the execution time e_N and power consumption p_N, i.e., $j_N = e_N p_N$.

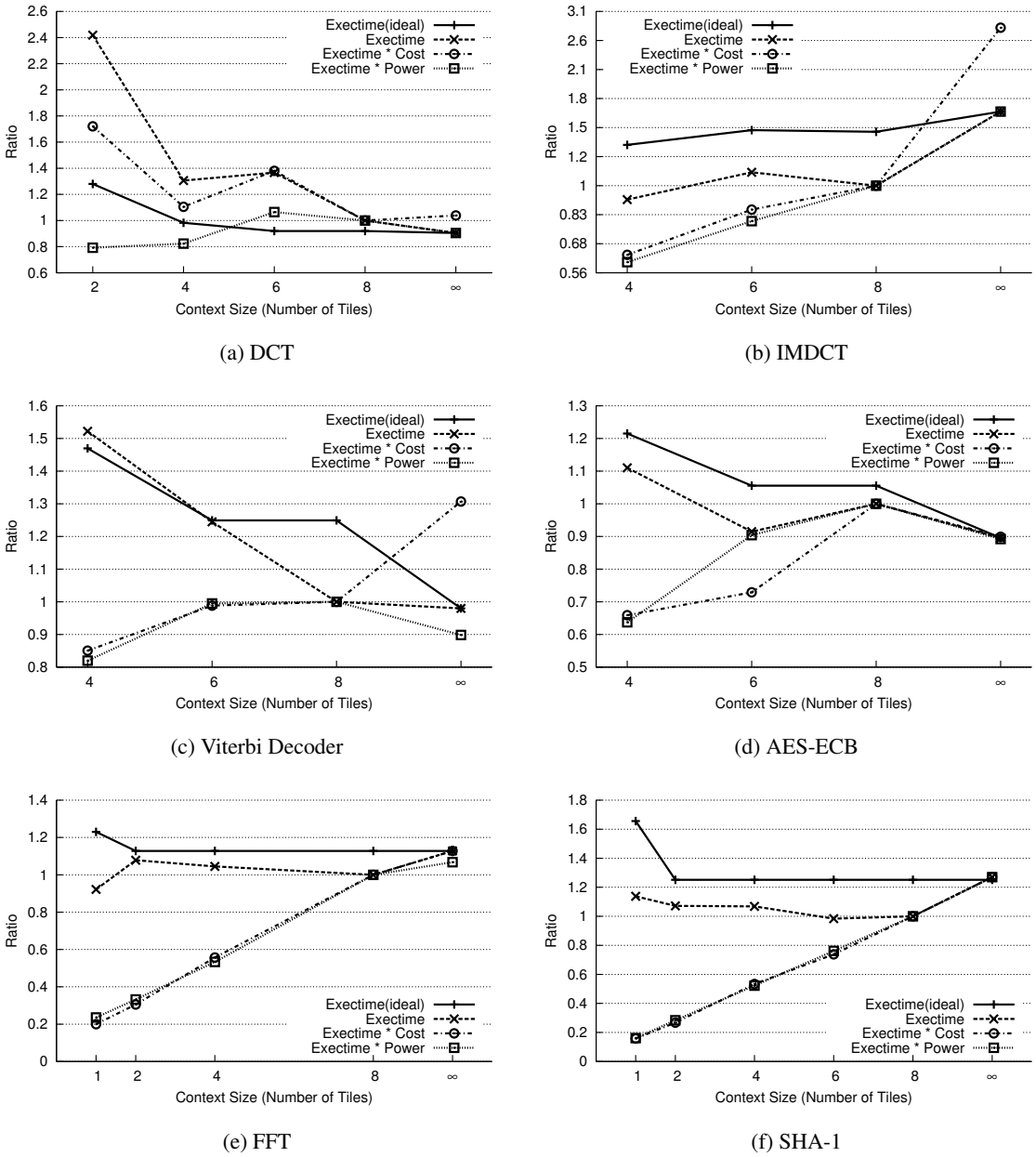
4.6.2 Trade-off Evaluation Results

The evaluation results for each benchmark application are shown in Fig. 4.6. In these figures, each result is normalized to the result in case of 8-Tile ($N = 512$). In addition, the execution time e_N, cost-performance ratio r_N, and required energy consumption j_N are represented by Exectime, Exectime * Cost, and Exectime * Power, respectively.

From the evaluation results for execution time, we can see that a large context size can improve the performance of all of the applications as long as the parallelism is retained. In particular, in the case of highly-parallel applications, since the execution clock cycles are extremely reduced with the increasing context sizes, the performance improvement is significant. On contrast, the performance of sequential applications such as FFT and SHA-1 cannot be improved even with more than one or two Tiles as the context size. This is because that execution clock cycles cannot be so reduced with the increasing context sizes.

We compare with ideal execution time derived from the estimation method (as denoted by Exectime (Ideal)) and actual measurement. As mentioned above, if the overhead caused by the step divisions and multiple-step allocations is exposed, there exists a great difference between ideal execution time and real execution time. However, by using the estimation method, we can predict the number of Tiles in which the limitation of the performance improvement exists. For example, in the case of DCT, the ideal execution time estimated by the model remains unchanged with more than 4 Tiles. The actual measurement shows that the execution time can be reduced much when increasing from 2 Tiles to 4 Tiles, but having more than 4 Tiles cannot lead to additional performance improvement, just as the model estimated. On the contrary, as shown in the comparison between the cases of 4-Tile and 6-Tile DCT designs, the delay time enlargement can compromise the performance.

In Viterbi, the ideal estimation does not reveal the significant effect to performance between 6-Tile and 8-Tile designs, but the performance improvement can be expected with more than 8 Tiles



(a) DCT

(b) IMDCT

(c) Viterbi Decoder

(d) AES-ECB

(e) FFT

(f) SHA-1

Fig. 4.6: Performance, Area Cost, and Power Consumption vs. Context Size

as the context size N. Actually, since the performance of 6-Tile Viterbi design is improved by the increase to 8 Tiles, we can say that Viterbi is a highly-parallel application which can be accelerated by increasing the context size to more than 8 Tiles. For AES-ECB, similar to Viterbi, more than 8-Tile context size is expected to have a great advantage for the performance improvement. However, since the improvement possibility is not so high, we can conclude that the performance improvement exists for 6-Tile as estimated by the model.

For area-efficiency or cost-performance, in the case of DCT, although the 6-Tile design is inefficient because of the performance degradation, the area-efficiency can be improved by increasing the context size to 8 Tiles. For the other applications, the evaluation result shows that the design with the smallest context size is most area-efficient. This is because the reduction of execution time is exceeded by the increase of area cost with increasing context size. The same result can be also shown in the ideal estimation of the model.

For power-efficiency, similar to the results of area-efficiency, the smallest context size leads to the smallest required energy consumption, i.e., most power-efficient, for all benchmark applications. In the case of DCT, although the 4-Tile design has a great advantage of performance improvement compared with the 2-Tile design, because of the higher increase ratio of the power consumption, the required energy consumption in the 2-Tile design becomes smallest.

Throughout the trade-off evaluation, we can claim that there exists a correlation between area- and power-efficiency. Therefore, by selecting the context size in which the cost-performance ratio is best, the energy consumption can be also optimized for most applications.

4.7 Summary

In this chapter, the context-level time-multiplexed execution model for dynamically reconfigurable processors was described, and the fundamental trade-offs between performance, area cost, and power consumptions for various context sizes were discussed. For performance and area cost, the parallelism of target applications were evaluated assuming the infinite number of available PEs ($N = \infty$). Then, the execution time and area cost for each context size were modeled based on the parallelism.

We have implemented several benchmark applications onto NEC Electronics' DRP-1 and verified the validity of the model. Each application has been designed for different context sizes, and the optimal context size for area- and power-efficiency has been evaluated. As a result, although the model can estimate the trend of varying performance and area cost for each context size, there exist significant differences between the results from the model and the actual measurement. The main cause can be guessed that the effect of step divisions and multiple-step allocations on the delay time and the overhead caused by the step divisions are difficult to be estimated by the model.

The trade-off evaluation results reveal that there is a correlation between area/power-efficiency and context size. Therefore, the context size which optimizes the area-efficiency can also optimize the power-efficiency for many applications.

From the evaluation results, we suggest the following methodology to find the optimal context size for the target application.

1. Evaluate the parallelism of the target application assuming the infinite context size $N = \infty$,
2. Estimate the execution time by using the model described in this chapter, and
3. Choose the smallest context size (the number of Tiles) which can achieve required execution time for the optimal area- and power-efficiency.

In this work, however, we have evaluated the restricted variety of the benchmark applications and the data width of PE which has been fixed to 8-bit. Future work should include the evaluations of various architectural trade-offs. In contrast, for FPGA architectures, a wide variety of evaluation results have been reported. Recently, more power-efficient architecture has been already investigated, and FPGA vendors have released the power-aware products such as Xilinx Spartan FPGAs as shown in Section 2.1.4. We believe that the quantitative evaluations and comparisons to such low-power FPGAs are required for more power-efficient dynamically reconfigurable processors.

In addition, in order to estimate the optimal context size in area and power consumption, an analytical model which considered only application parallelism was used in this work. For more accurate cost and performance model for context-level time-multiplexed execution, we can guess that the application characteristics including communication patterns between PEs and memory element access patterns have to be considered. However, since these characteristics highly depend on the architecture or device, their considerations will increase the complexity of the model very much.

Chapter 5

Task-level Time-multiplexed Execution Model

In this chapter, a task-level time-multiplexed execution model is described. The first section describes an overview of this chapter, and the following three sections introduce practical application examples mainly investigated by the author.

5.1 Overview

We investigate the task-level time-multiplexed execution model taking practical applications into account. In the task-level time-multiplexed execution model, a task division technique and a task scheduling method are significant research topics. However, individual tasks in a stream-based processing have a relatively strict independence, and the task flow is almost rectilinear [83]. In this thesis, we evaluate the task-level time-multiplexed execution model through developing practical application examples. The task scheduling and mapping techniques are being investigated by our colleague [84] and they are out of the scope of this thesis.

We have proposed various applications by using NEC Electronics' DRP-1. This chapter explores three applications of the task-level time-multiplexed execution model. Similar to the context switching control, there exist dynamic and static control schemes for switching task. However, since most multimedia applications have a straight task flow, a complicated and flexible dynamic control is not needed. In this chapter, special applications which require a dynamic task switching control such as data-driven and event-driven controls are experimented. In contrast, a simple static task switching control scheme is implemented and evaluated for MuCCRA-1 (see Chapter 6).

At first, Section 5.2 describes a DRP-based cryptographic accelerator which supports flexible cryptographic task switching for mobile Internet communications. This accelerator provides a data-driven control scheme for switching the cryptographic tasks. The task indicating a packet header can be executed on a single DRP core. Throughout this work, we evaluate the time overhead caused by cryptographic task switching and its reduction effect due to background task loading.

Secondly, Section 5.3 explores an adaptive computing which can dynamically optimize performance and power consumption trade-offs depending on operational situations such as a signal noise

for mobile wireless communications. With the task-level time-multiplexed execution, it is feasible that power consumption can be saved by switching a performance-intensive task and a power-intensive task on a DRP core depending on the operational situation. We apply the adaptive computing to an error correction algorithm, Viterbi, for mobile wireless communications. The adaptive Viterbi decoder has an event-driven control mechanism in which Viterbi decoder tasks can be switched depending on a signal-to-noise ratio. By investigating the adaptive Viterbi decoder, a power reduction effect due to the task-level time-multiplexed execution is analyzed.

Finally, a multi-core architecture with multiple DRP cores is investigated in Section 5.4. Multiple cores of the dynamically reconfigurable processor are connected to each other via on-chip networks. In this architecture, an application can be executed with minimized inter-core communications by performing the task-level time-multiplexed execution on each core. The configuration data corresponding to each task has to be transferred to each core. In this work, we evaluate the performance of the on-chip network for various traffic patterns.

5.2 Data-driven Virtual Hardware: IPsec Accelerator

5.2.1 Motivation and IPsec Overview

This section describes a DRP-based cryptographic accelerator for IPsec [12, 13]. IP Security (IPsec) [85] is a protocol suite that provides a security feature to the standard Internet Protocol (IP) with a combination of multiple encryptions and authentications. It has also become popular in embedded systems with a widespread of mobile terminals. Since powerful computational capability is required to deal with data encryption and authentication processing, hardware implementation has been a common solution. However, designing customized cryptographic engines results in a heavy increase in development cost. In addition, it is difficult to cope with newly developed or standardized cryptographic algorithms.

In this work, a target system has a single DRP core as a coprocessor of an embedded microprocessor for accelerating cryptographic tasks. The DRP core supports several cryptographic tasks used in IPsec, and they can be dynamically switched depending on an IP-packet header by exploiting the task-level time-multiplexed execution.

5.2.2 System Design Policy

At first, the target system architecture of the cryptographic accelerator is introduced.

(1) Tightly Coupled Coprocessing System

As mentioned in Section 2.2, a reconfigurable fabric including the dynamically reconfigurable processor can be classified into three types of system components. In the case of the cryptographic accelerator, a tightly coupled coprocessor system shown in Fig. 5.1 is suitable. It consists of an embedded

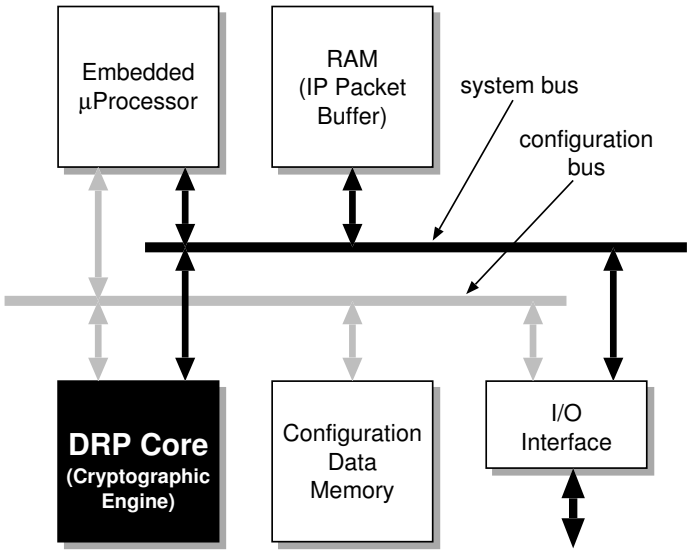


Fig. 5.1: Target System Architecture

microprocessor for control-intensive tasks, a DRP core as a coprocessor for cryptographic tasks, and several memory modules for IP packets and configuration data. In this system, each cryptographic task can be executed on the DRP core with less inter-task communications. If the DRP core is used as a reconfigurable functional unit of the microprocessor, frequent data transfers between the DRP core and a register file of the microprocessor occur, and it can be a crucial performance bottleneck of the system.

The embedded processor, DRP core, and shared RAMs including an IP-packet buffer and configuration data memory are connected by high-speed on-chip buses like NECoBus [86], and a DMA transfer is supported. The IP packets to be processed are stored in the IP-packet buffer, and cryptographic tasks are in the configuration data memory. In this system, each cryptographic task is individually implemented on the DRP core.

(2) Background Configuration Data Transfer based on Double Buffer

In this system, by exploiting the task-level time-multiplexed execution model, more and more cryptographic tasks can be virtually supported as far as the capacity of the configuration data memory allows. We refer to this kind of systems as *virtual hardware*. In the virtual hardware system, the time for configuration data transfers can be a bottleneck of the system. Since task switching may be frequently occurred when many nodes communicate using several cryptographic tasks on IPsec, the time overhead caused by the configuration data transfer is a critical problem in the cryptographic accelerator.

To address this problem, the background configuration data transfer scheme based on double-buffered context memories is introduced. In this method, at first, available contexts of the DRP core are divided into two groups. Then, a certain task in one context group is executed, while the

configuration data of the next task is transferred to the other context group. Consequently, the task execution and the configuration data transfer can be overlapped, and the configuration data transfer is completed in background. In order to completely hide the time overhead of the configuration data transfer, however, the next cryptographic task must be predictable. In addition, the next configuration data transfer must be finished during the execution of the current task to switch the task without disturbing of system operations.

(3) Acceleration Flow

The acceleration flow of the cryptographic accelerator is as follows. At first, the embedded microprocessor checks an IP-packet header in the IP-packet buffer. It determines whether IPsec is applied for the packet. In IPsec, two communicating parties preliminarily negotiate and establish a security association (SA) for protecting the transferred data. An SA specifies the cryptographic algorithm and the related keys to be utilized. The embedded microprocessor decodes the IP-packet header and finds out the corresponding SA from its database called security association database (SAD). After that, it searches the configuration data corresponding to the cryptographic task indicated by the SA and sends it to the DRP core.

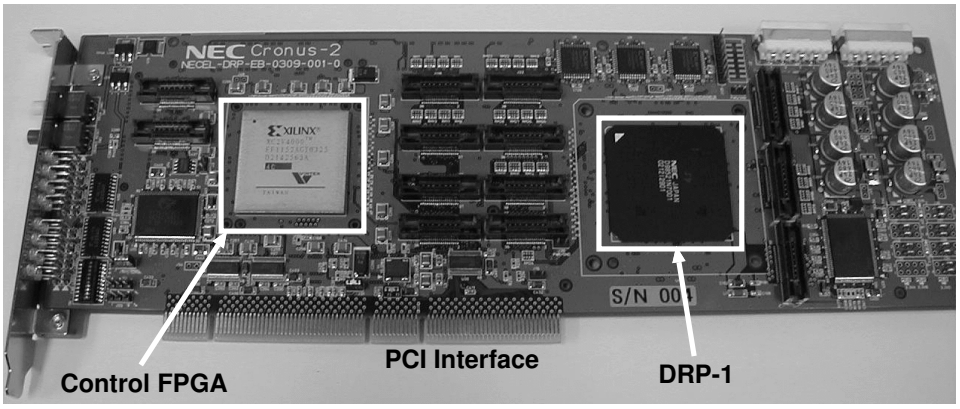
For example, if an SA indicates DES-CBC as an encryption of an IP-packet payload, the configuration data is transferred to the DRP core according to a pointer from the embedded microprocessor. After configuring the DRP core, the embedded microprocessor sends the data to an input buffer of the DRP core. Then, it starts processing of DES-CBC. The output data is written in an output buffer, and the embedded processor pulls it. Finally, an IPsec packet is transferred to a network interface or an upper-layer application.

For the background configuration data transfer based on the double buffer, the embedded microprocessor looks ahead the IP-packet buffer and generates an event sequence of the future task transition. These features lead to overlap the configuration data transfer for the next task with the current task execution.

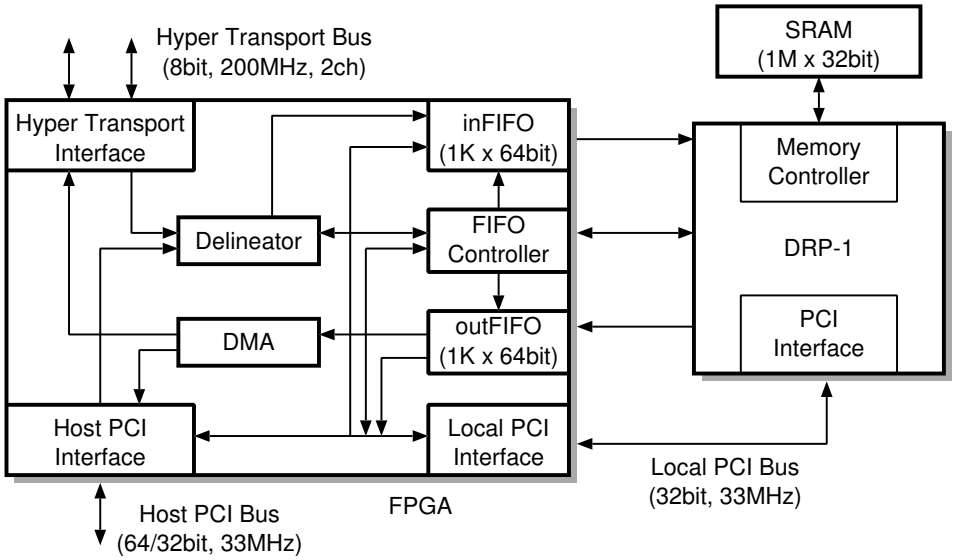
5.2.3 Experimental System

(1) DRP Evaluation Board

For verifying the cryptographic accelerator, we established the experimental emulation system by using a DRP evaluation board shown in Fig. 5.2. This is a PCI board consisting of a DRP-1, an FPGA as I/O controller, a PCI interface, and an external SRAM. In this experimental environment, the DRP-1 is used as a DRP core with 8 Tiles and a host PC controls the DRP-1 using a set of application programming interfaces (APIs) instead of the embedded microprocessor. We developed and debugged the cryptographic accelerator on the environment with the host PC and the DRP-1. We use this evaluation board as an emulation platform though it does not support background configuration data transfer at the moment.



(a) Board Photo



(b) Block Diagram

Fig. 5.2: DRP Evaluation Board

(2) Cryptographic Task Designs

The cryptographic tasks implemented on DRP-1 are five private-key encryptions and decryptions including DES, AES (Rijndael), CAST-128, CAST-256, and RC6. Two one-way hash functions: MD5 and SHA-1 have been also implemented. They have been described in the C-level description language, BDL as introduced in Section 3.3.2, and compiled with the DRP compiler. The Cipher Block Chaining (CBC) mode, which is generally used in IPsec, is adopted as an operation mode for all of the private-key encryption algorithms.

The DRP compiler supports two context scheduling methods, automatic and manual scheduling. The manual scheduling in which context switching is directed manually by specifying timing directives in a BDL code, while the automatic scheduling mode does not need it. For the fully optimized context scheduling, we adopted the manual scheduling.

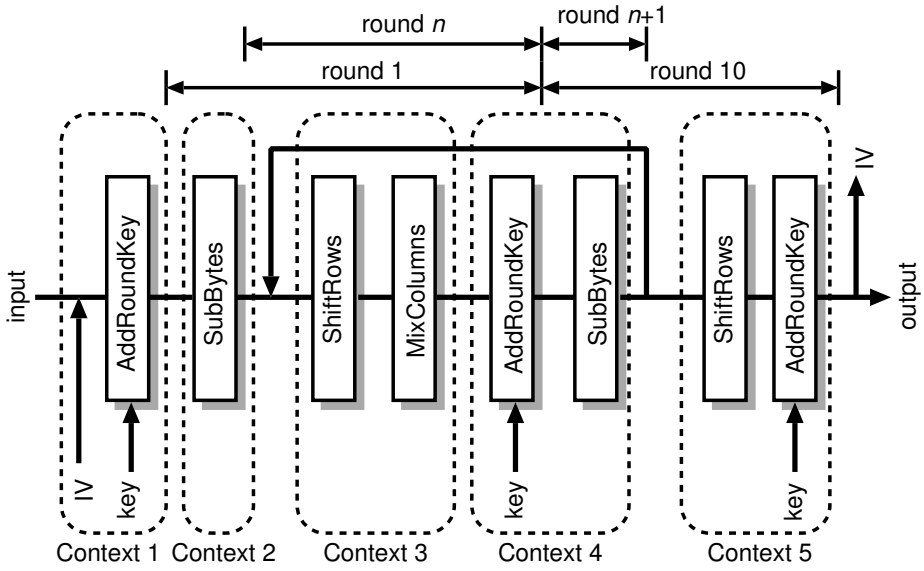


Fig. 5.3: AES Context Scheduling for DRP-1

There are various task distributions between the embedded microprocessor and the DRP-1. Key schedulers for DES and AES are also supported on the DRP-1, while software supports the other algorithms. The AES, CAST-256, and RC6 are designed to be able to select a key-length flexibly. The MD5 and SHA-1 are designed to calculate a hash value of an original message by 512-bit blocks. The other operation of MD5 and SHA-1, such as zero-padding and appending a message length are done by software. Furthermore, by processing with software and the DRP-1, it is possible to compute an HMAC [87] which is an authentication algorithm used in IPsec.

(3) Case Design: AES-CBC

As an example of the implementations of cryptographic tasks, only the AES-CBC design is described in this section because of space limitation.

Advanced Encryption Standard (AES) [88] is a 128-bit block cipher which has been selected as a successor to the venerable Data Encryption Standard (DES) by the National Institute of Standards and Technology (NIST). The AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits. The AES is designed to use only simple whole-byte operations. In the case of 128-bit key, AES iterates 4 steps for 10 rounds. Each regular round involves four steps: SubBytes, ShiftRows, MixColumns, and AddRoundKey. MixColumns step is skipped in the final round. In the CBC mode, expanded keys are used in each round, and the 128-bit cipher block is fed back to the next encryption as Initialization Vector (IV).

The context scheduling of AES-CBC is shown in Fig. 5.3. Since AES uses only 8-bit operations, the DRP core deals with it well due to the granularity of the PE and exploit the most of the byte-level parallelism especially in MixColumns step. First is the SubBytes step, where each byte of the block

Table 5.1: Resource Usage and Throughput of Cryptographic Tasks

Task	Required Resources				Freq.	Throughput[Mb/s]		
	Ctxt	Max	All	Vm	[MHz]	DRP-1	DSP	MIPS
DES	16	121	1357	10	28.6	107.6	84.2	40.3
CAST128	9	32	140	20	27.4	109.8	95.4	46.7
AES	8	128	372	30	56.7	363.1	16.9	46.4
CAST256	16	38	327	21	28.9	37.4	36.1	23.2
RC6	6	56	164	8	34.0	198.1	96.6	76.9
MD5	7	75	279	14	24.7	194.2	135.5	95.5
SHA-1	8	65	337	20	35.1	202.1	44.0	38.7

is replaced by its substitute called an S-box operation. We implemented the S-Box using VMEMs as LUTs to achieve a balanced design in resource and delay. Since the VMEM is a 2-ported memory and distributed around a Tile of the DRP core, required data sets can be read from the VMEM-based S-box simultaneously.

To hide the 1-clock latency for the memory access, round n and round $n+1$ are mixed in the same context as shown in Fig. 5.3. This causes inefficient resource utilization since the same functions are required in the different contexts. The number of required contexts is increased with the performance improvement.

5.2.4 Cryptographic Task Evaluations

(1) Resource Usage

Table 5.1 shows resource usage and throughput of each cryptographic task. The required contexts (Ctxt), the maximum number of required PEs (Max), the total utilization of PEs (All), and the maximum number of required VMEM modules (Vm) are summarized in this table. The value of Max is the maximum number of PEs used among contexts. This implies the number of PEs used spatially. In contrast, the value of All means the total number of PEs used in all contexts, i.e. the number of PEs used temporally.

One Tile of DRP-1 has 64 PEs. So, all of the cryptographic tasks are implemented within 2 Tiles of DRP-1 (128 PEs and 32 VMEMs). We can optimize the area-efficiency by adopting a DRP core with 2 Tiles for all the cryptographic tasks. The context which utilizes the largest number of PEs includes a core round function of the encryption process, which is the critical path of all contexts.

(2) Throughput

We have also evaluated the throughput of each cryptographic task, with the result summarized in Table 5.1. The throughput has been measured with the minimum number of clock cycles which is

Table 5.2: Configuration Data Size of Cryptographic Tasks

Task	Config. Data Size [bit]
DES (Encryption + Key-Scheduler)	331104
CAST128 (Encryption)	104928
AES (Encryption + Key-Scheduler)	201184
CAST256 (Encryption)	207744
RC6 (Encryption)	90240
MD5	113760
SHA-1	134560

required to operate 10000 blocks. The throughput on the DRP-1 was compared with the Texas Instruments DSP which is the same as the one used in Chapter 4. NEC's MIPS64 compatible VR5500 was also used for the evaluation. The VR5500 is a 10-stage pipeline, 2-way out-of-order SuperScaler processor that runs at 400MHz with 32-KB instruction cache and 32-KB data cache. Evaluation programs for each cryptographic task were written in C language and compiled with a MIPS compatible gcc cross-compiler in the T-Engine design environment. The '-O3' optimization option was used for compiling the programs.

Evaluation result shows that the throughput of all the cryptographic tasks on the DRP-1 outperforms the VR5500 from 1.6 to 7.8 times. So, they can accelerate encryptions and authentications for embedded IPsec communications. Because the performance is better than that of the DSP, the DRP-1 has an advantage as an accelerator. In terms of resource usage, all cryptographic tasks have been implemented within 2 Tiles of DRP-1. If we use 2 Tiles as a DRP core, it is possible to accelerate the cryptographic tasks with improved area-efficiency. Furthermore, since the operational frequency is about 30MHz, the consumed power is expected to be reduced compared to other architectures.

5.2.5 Analysis of Run-time Configuration Data Transfer

(1) Overhead of Configuration Data Transfer

The implementation results show that the DRP core has potential to accelerate each cryptographic task compared to the DSP or the embedded microprocessor. In the proposed system, however, the cryptographic tasks are dynamically switched in the task-level time-multiplexed execution manner. Namely, they are switched by replacing the configuration data corresponding to each task on demand. Thus, the run-time reconfiguration overhead in the task switching must be evaluated.

The size of configuration data of each cryptographic task is shown in Table 5.2. About 40KB of the configuration data in a large case must be transferred from the configuration data memory into the DRP core. Although it is much smaller than the configuration data of common FPGAs, its reconfiguration time is considerable compared with the execution time.

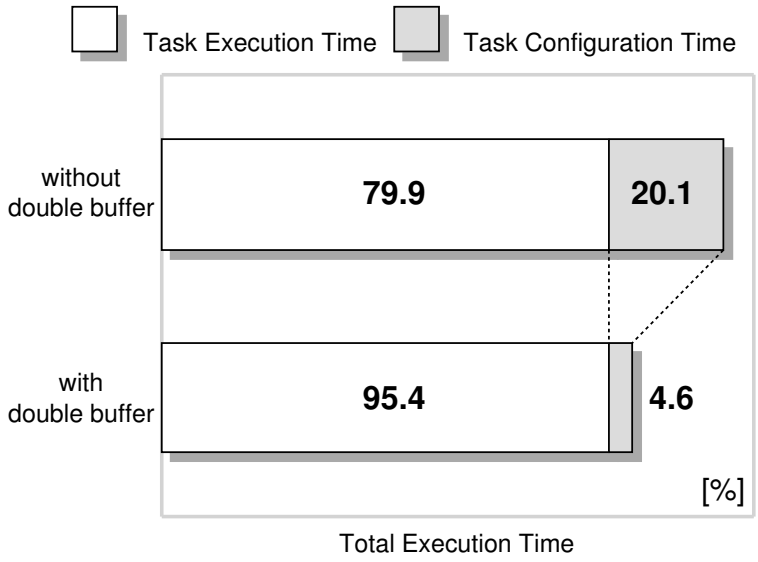


Fig. 5.4: Run-Time Reconfiguration Overhead and Impact of Background Transfer

(2) Background Transfer Impact

We simulate the virtual hardware system with the background configuration data transfer mechanism using an actual IPsec traffic and evaluate the effect of reducing the overhead. Each cryptographic task has been implemented within 16 contexts of the DRP-1. Thus, we suppose that 32 contexts are available to allow using the double-buffered context memory. It is also assumed that the bandwidth for transferring configuration data is 128bit/100MHz. The configuration access is controlled by the on-chip embedded microprocessor.

The traffic data of IPsec is traced from the following actual network with three personal computers (PC-A, -B, and -C) communicating with each other using IPsec. PC-B and -C download a file of a certain size from PC-A simultaneously. The traffic data streaming in the above network is measured. PC-A and -B communicate by using encryption by AES-CBC and authentication by HMAC-MD5-96. Similarly, the PC-A and -C communicate by using RC6-CBC and HMAC-SHA-1-96. It is assumed that the cryptographic accelerator is mounted in the PC-A, which is a host PC in this network. In addition, we suppose that enough size of packets are stored in the IP-packet buffer of the PC-A to predict and prefetch the next configuration data during the execution of a certain task.

Fig. 5.4 shows the simulation result of the overhead caused by configuration data transfers. In this graph, the ratio of required transfer time of tasks in the case of with/without the double-buffered context memory is shown. If the configuration data transfer cannot be overlapped, the overhead reaches at 20.1% of the total operation time. If the double-buffered context memory is available, the overhead decreases to 4.6% and it is possible to reduce 80.7% of the transfer time compared with the case where no transfers can be hidden, i.e., transfers cannot be done in the background.

The effects of background configuration data transfers are classified broadly into the following two categories.

- One is the effect of hiding the transfer time behind a task execution.
- The other effect is the reduction of configuration frequency due to buffering of tasks in the DRP core. In this simulation, since 19.7% of the tasks were retained in the DRP core, their configuration data transfers were skipped.

From above analysis, we have demonstrated that the overhead can be reduced to 4.6% of the total operation time. Consequently, it is also possible to implement a practical virtual hardware system with a DRP core. In addition, if more contexts (e.g. 64 contexts) are available, more tasks can be supported, and it is easy to reduce the overhead. The hierarchical configuration cache can improve the bandwidth of the run-time configuration data transfers.

5.2.6 Design Comparison

Recently, FPGA-based cryptographic accelerators have been developed. By exploiting one or more FPGAs, high-performance and flexible systems have been implemented.

In [89], SLAAC-1V board that has three Xilinx's Virtex XCV1000 has been proposed. High-throughput Triple-DES and AES are implemented on this accelerator. This board is a powerful and large scale cryptographic accelerator based on multiple FPGAs.

A task switching method with run-time FPGA reconfiguration has been introduced in [90]. In this literature, Adaptive Cryptographic Engine (ACE), which is an IPsec accelerator based on a single FPGA is proposed. Five private-key encryptions are implemented, and ACE can switch them by dynamically reconfiguring the FPGA. For the challenge to the configuration data transfer overhead, an efficient use of a configuration data memory by compressing the configuration data is examined.

These FPGA-based systems achieve high throughput and flexibility, but require multiple high-end FPGAs as attached reconfigurable processing units which are connected to host PC by an interface like PCI and USB. These systems are for acceleration of common PCs and difficult to be integrated in embedded systems. In addition, millisecond-order reconfiguration time of common FPGAs is not actually acceptable for embedded systems. The proposed cryptographic accelerator for IPsec is advantageous compared with conventional approaches due to the flexible task-level time-multiplexed execution.

5.2.7 Summary

In this section, the cryptographic accelerator for IPsec based on NEC Electronics' DRP was presented. This system accelerates multiple cryptographic tasks on the DRP core and also can switch them on demand in the task-level time-multiplexed execution manner.

The evaluation result shows that all of the cryptographic tasks can be implemented within 2 Tiles of DRP-1. The throughput of each cryptographic task outperformed a DSP and MIPS64-compatible microprocessor. In addition, we analyzed the overhead of the run-time configuration data transfer.

The evaluation result shows that the overhead is 20.1% of the total operation time. We were able to reduce about 80.7% of the overhead by background configuration data transfers.

In recent years, there is a strong trend toward multi-standard or multi-application SoCs. For example, recent video recorders support older video compression standards, e.g. MPEG-2 as well as newer standards such as MPEG-4 or WMV9 together with H.264. Recent mobile handsets also support multiple wireless standards such as IEEE 802.11x, Bluetooth, and Ultra-WideBand (UWB). The proposed system architecture has a great advantage in flexibility and cost-efficiency for these multi-standard or multi-application products compared to conventional customized ASIC designs.

5.3 Event-driven Adaptive Viterbi Decoder

5.3.1 Motivation

In this section, the other experiment of the task-level time-multiplexed execution model is described in order to demonstrate the applicability to battery-aware mobile products.

Mobile computing systems, powered for hours or days by batteries, are required to suppress energy consumption, maintaining quality of service (QoS). Especially in mobile communication, it is required to maintain a certain bit error rate (BER) while signal-to-noise ratio (SNR) on a channel varies. The system should optimize its error correcting capability to save energy in response to the variation of SNR.

Adaptive computing [91, 92] is a useful technique for power saving by making full advantage of reconfigurable devices. In this method, the system uses a set of multiple tasks which provides a same function with the different performance and power consumption. In response to a given condition, a reconfigurable system or device changes its configuration so as to fit the condition and improve performance and energy-efficiency.

The energy and time effectiveness of the adaptive computing during reconfiguration depends on the device. Although the adaptive computing on a traditional FPGA has been reported in [91], it spends long reconfiguration time and extra power consumption for loading configuration data. The long reconfiguration time of traditional FPGAs leads to a temporal growth of the latency which can be fatal in communication. In the task-level time-multiplexed execution on the dynamically reconfigurable processors, the reconfiguration time and energy consumption are much smaller than those of FPGAs, and the barrier for introducing the adaptive computing is much reduced.

In this section, design and implementation of the adaptive Viterbi decoder [14, 15] is described. We apply the adaptive computing scheme to Viterbi decoder, which is the maximum likelihood decoding algorithm for a class of error correcting technique known as convolutional coding. By switching tasks of the Viterbi decoder flexibly based on the task-level time-multiplexed execution model, the average performance and energy consumption can be optimized.

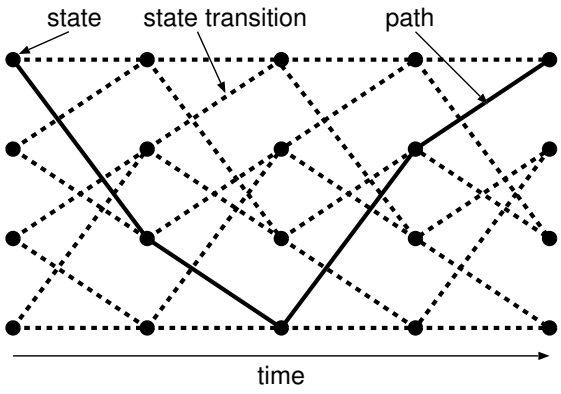


Fig. 5.5: Trellis Diagram ($K = 3$)

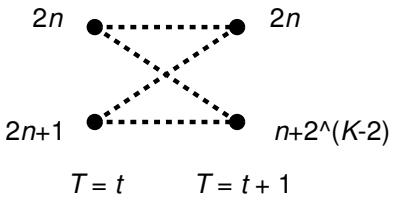


Fig. 5.6: Branch Metric

5.3.2 Viterbi Decoder Overview

Error correction coding can be used to detect and correct data transmission errors in wireless communication channels. Encoding is accomplished through the addition of redundant bits to transmitted information symbols. These redundant bits provide decoders with the capability to correct transmission errors. Convolutional codes form a set of popular error-correction codes. In convolutional coding, the encoded output of a transmitter (encoder) depends not only on the set of encoder inputs received during a particular time step, but also on the set of inputs received within a previous span of $K - 1$ time units, where $K \geq 1$. The parameter K is the constraint length of the code.

The Viterbi algorithm, which is the most extensively employed decoding algorithm for convolutional codes, works well for less-complex codes, indicated by the constraint length K. However, the algorithm's memory requirement and computation count pose a performance obstacle when decoding more powerful codes with large constraint lengths.

(1) Add-Compare-Select (ACS) Unit

The behavior of a Viterbi decoder is often represented with a trellis diagram, as shown in Fig. 5.5. The trellis is a kind of state transition diagram consisting of horizontal axis representing time (stage) and vertical axis for states which are formed with an input sequence. The number of states is 2^K where K is the constraint length. The function of the decoder is to attempt to restrict the input sequence by making state transition patterns or paths from the received sequence. To make a path, the decoder determines a cost called path metric at each state. As shown in Fig. 5.6, each state at $T = t$ may transit to two states at $T = t + 1$, and a cost called branch metric is calculated according to the direction and input value of the decoder. The path metric for the next state is calculated from the sum of path metric in the current state and branch metric. The state transition sequence with the smallest path metric is selected as a path. In common Viterbi decoders, the above process is done with a hardware module called Add-Compare-Select (ACS) unit. The amount of required calculations in the ACS increases with K.

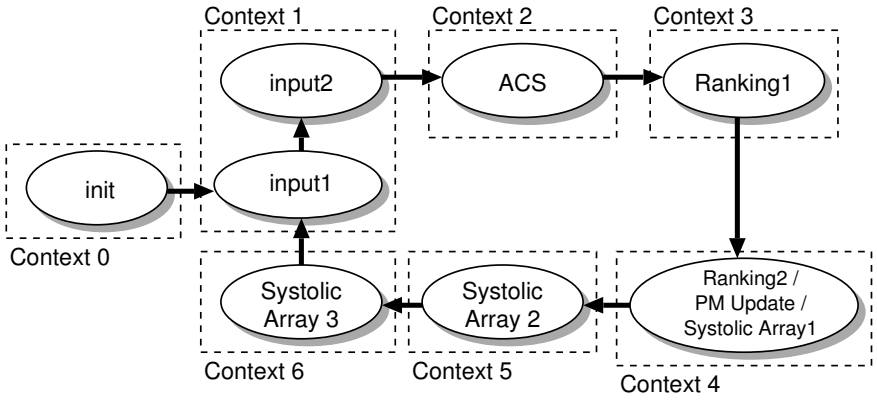


Fig. 5.7: Context Scheduling of Viterbi Decoder ($K = 7$)

(2) Trace Back Unit

The original bit-sequence is decided from the selected path with a trace back unit. Instead of a LIFO-based method used in software implementations, a systolic array [93] or Register Exchange Array (REA) [94] is commonly used in hardware implementations. Although REA can make the best use of parallelism, it tends to require too large hardware resources for mobile devices. Here, the systolic array is used for the trace back module.

In order to keep sufficient error correction capability, the length of the path (trace back length) should be $5(K-1)$. In this case, the stage number of the pipelined systolic array becomes $5(K-1)$.

5.3.3 Adaptive Viterbi Decoder Design

We focus on a constraint length K for introducing the adaptive computing to the Viterbi decoder. The constraint length K determines the size of a trellis diagram and it is defined by

$$(\text{the number of states } 2^{K-1}) \times (\text{truncation length } 5K).$$

The error-correcting capability of a convolution is improved by employing a large size of trellis diagram with a large K. On the other hand, the complexities of both the ACS and trace back units are increased with a large K. Increasing K leads to an exponential growth in the amount of computation and retained path storage. It also increases the power consumption and decreases the throughput.

In the adaptive Viterbi decoder, several decoder designs corresponding to different K are provided, and changed according to the SNR. For a severe condition with low degree of SNR, a large sized design with a large K is used to provide enough error correction capacity. When the SNR is improved, that is, the condition of wireless communication becomes good, the power consumption can be saved by using the design with a small K. This implies that the decoder can be dynamically switched so as to optimize the power consumption keeping the required error correction capacity.

We have implemented five Viterbi decoders with constraint length $K = 3, 4, 5, 6, 7$ on the DRP-1. The other parameters are set as follows: coding rate is $1/2$, trace back length is $5(K-1)$, and

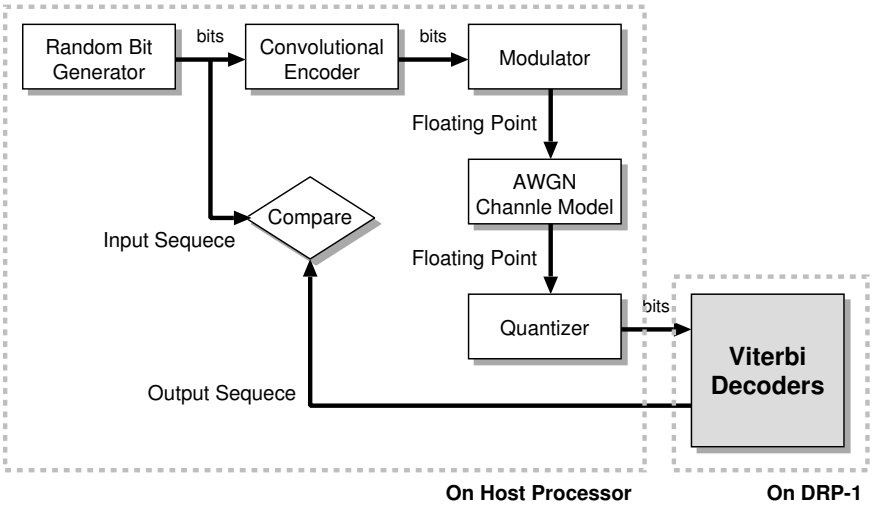


Fig. 5.8: The Simulation Model for Adaptive Viterbi Decoder

input/output width is 1 bit. The Viterbi decoding was scheduled onto contexts as shown in Fig. 5.7. The figure is for $K = 7$, and those for smaller K are simpler. After data input in Context 1, the ACS is performed in Context 2. A decoder with a constraint length K has 2^{K-1} ACS units, but all of them can be implemented on a context even when $K = 7$. Although the number of contexts is same, Context 2 for $K = 7$ uses much more PEs than that for $K = 3$, and requires more power consumption.

The trace back unit with a constraint length K has $5(K - 1)$ pipeline stages for its systolic array structure. Since 30 stages are required for the design with $K = 7$, they are divided and implemented on three contexts (Context 4, 5 and 6) as shown in Fig. 5.7. The trace back unit for $K = 6$ that requires 25 stages is implemented on two contexts. In other designs with smaller $K = 3, 4, 5$, every operation corresponding to Context 4, 5, and 6 in Fig. 5.7 can be mapped on only a single context, and the required number of contexts can be reduced.

5.3.4 Evaluation Results

We have implemented a simulation environment of the adaptive Viterbi decoder. This environment is subjected to investigate various features of the designed Viterbi decoders including error correction capability and power consumption. For this purpose, we have developed a simulation program on a personal computer with Linux kernel 2.4.31 as shown in Fig. 5.8. And also, this environment has a DRP evaluation board as same as the cryptographic accelerator shown in Section 5.2. The received bit sequences including a certain bit-errors are generated with the software on the computer, transferred to the DRP-1, and decoded. The results are returned to the computer and checked.

(1) Performance Summary of Viterbi Decoders

The results are summarized in Table 5.3. The Viterbi decoder implemented on the DRP-1 with a larger K has more powerful error-correcting capability, more contexts required, higher power con-

Table 5.3: Performance Summary of Viterbi Decoders

Constraint Length K	3	4	5	6	7
SNR ($BER = 10^{-5}$)	5.3	4.9	4.3	3.8	3.2
# of Contexts	2	4	4	5	7
# of Clocks (Clocks/Symbol)	4	5	5	6	7
Critical Path Delay [ns]	25127	22340	26989	27390	30346
Maximum Operational Freq. [MHz]	39.8	44.7	37.0	36.5	32.9
Power Consumption [mW]	910.5	1061.3	938.6	1025.1	1029.0
Throughput [Mb/s]	9.95	8.95	7.41	6.08	**4.71**
Frequency at 4.71Mb/s (MHz)	18.9	23.6	23.6	28.3	33.0
Power Consumption at 4.71Mb/s (mW)	429.0	558.1	596.3	793.1	1029.0

sumption, and lower throughput than that with a smaller K. This table shows that the maximum throughput is changed from 4.71Mb/s ($K = 7$) to 9.95Mb/s ($K = 3$).

For comparing the absolute performance, we implemented the same Viterbi decoder on the NEC MIPS64-compatible VR5500. It is a 10-stage pipeline, 2-way out-of-order SuperScaler processor that runs at 400MHz with a 32-KB instruction cache and a 32-KB data cache. The throughput of the decoder with $K = 7$ on the VR5500 processor was 377 Kbits/symbol (10598 clocks/symbol), much smaller than those of the Viterbi decoders on the DRP-1.

Table 5.3 also shows the power consumption for achieving a fixed throughput 4.71Mb/s. The power consumption with $K = 7$ becomes more than double as that with $K = 3$ under this condition. From this table, it appears that the power can be saved by using designs with smaller K.

(2) Adaptivity Evaluation

We simulated mobile communication between a base station and a mobile terminal with a 30dB, 40dB, and 50dB gain. The simulation of the adaptive computing has been performed by varying the SNR of transmitted data and reconfiguring the ideal DRP based on K values required to achieve $BER = 10^{-5}$. The current DRP-1 provides 16 contexts, while the total contexts required for $K = 3, 4, 5, 6, 7$ becomes 23. In this evaluation, we assume the ideal case, and all contexts can be stored in the context memory of the DRP-1, and the decoder switching can be done with a clock cycle.

The DRP-1 is reconfigured once every 250,000 symbol. A set of 10^8 SNRs is generated using a log-distance path loss model with path loss exponent $n = 2.7$ and standard deviation $\sigma = 11.8$db [95] for the total transmission length of $250,000 \times 10^8$ bits[1].

Fig. 5.9 plots the variation in the power consumption at the fixed throughput (4.71Mb/s) for each gain. The horizontal axis corresponds to the distance between the base station and the mobile terminal. If the design with $K = 7$ is always used without the adaptive computing, the power consumption

[1]These parameters are based on the real measurement in German cities.

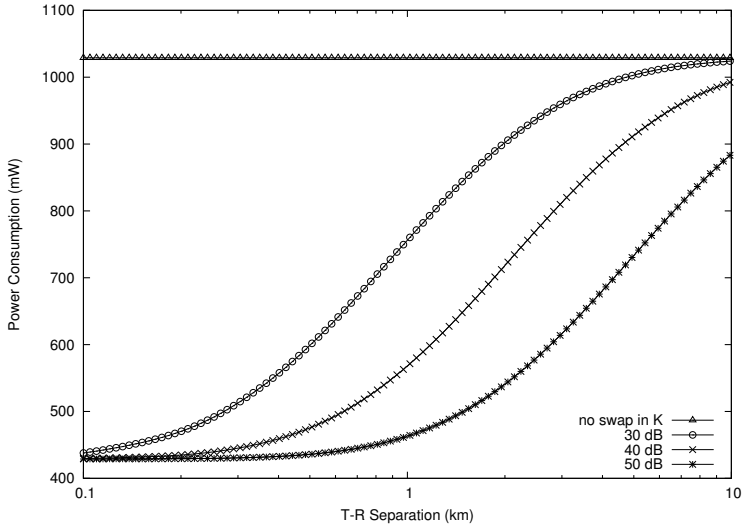


Fig. 5.9: Distance v.s Power Consumption at Fixed Throughput (4.71Mb/s)

Table 5.4: Configuration Data Size of Each Viterbi Decoder

K	3	4	5	6	7
Configuration Data Size [bit]	27968	45408	76480	128672	244032

is fixed independent from the distance. However, adaptive computing can save power consumption if the distance is not so long, which means the bit-error ratio is not so bad. If the distance is 0.6km and the gain is 50dB, power can be saved up to 58.3%.

Fig. 5.10 shows the variation in the throughput depending on the distance. In this case, the operational frequency is changed when the design is replaced so as to work at the maximum operational frequency of each design. Unlike the fixed throughput without the case of adaptive computing, the throughput is also improved with the adaptive computing if the distance is not so long. If the distance is 0.6km and the gain is 50dB, the throughput can be improved about 2.1 times.

5.3.5 Context-Memory Capacity Consideration

The above evaluation assumes enough capacity of the context memory to hold all Viterbi decoder designs for the task-level time-multiplexed execution. However, the DRP-1 has actually only 16 contexts, and 23 contexts are required to hold all Viterbi decoders. Thus, the virtual hardware mechanism is required in order to provide a full adaptivity to the Viterbi decoders. In addition, similar to the cryptographic accelerator described in Section 5.2, we would like to apply the background configuration data transfer scheme to the adaptive Viterbi decoder. At first, the configuration data size for each Viterbi decoder is derived and summarized in Table 5.4.

The task-level time-multiplexed execution model provides full flexibility for the designs to be selected on demand. However, if there is enough statistical information, we can select only frequently

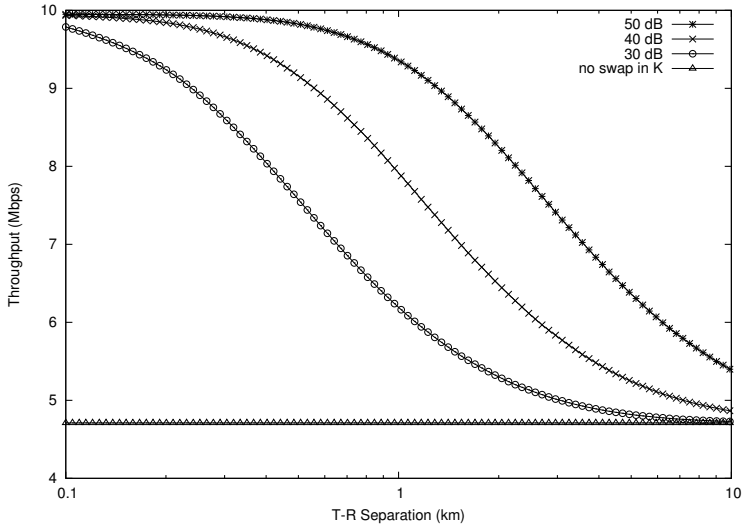


Fig. 5.10: Distance v.s Throughput at Maximum Operation Frequency

dispatched designs and make regular use of them retaining into the context memories statically. For this restricted adaptive computing, we can select the most frequently used Viterbi decoders with $K = 3, 5$ and 7.

For comparison, we evaluate the following four cases:

1. The design with $K = 7$ is always used without any task switching.
2. All designs with $K = 3, 4, 5, 6, 7$ are used assuming context memories to be unrestricted in capacity as same as the ideal case of Section (2).
3. The designs with $K = 3, 5, 7$ are always used without any configuration data transfers.
4. All designs with $K = 3, 4, 5, 6, 7$ are used, but the transfer from the configuration data memory is performed for every task switching.

The throughput and power comparisons are shown in Table 5.5 and Table 5.6. The results demonstrate that the performance of Case 4, which is based on virtual hardware, is almost equivalent to that of Case 2. This is due to the facts that (1) the coarse-grained PE array of the DRP-1 makes the configuration data transfer time from the configuration data memory relatively short, (2) coarse-timescale task switching (once every 250K symbols) suppresses the frequency of loading the configuration data, and (3) the system uses the designs with $K = 3$ and 7 frequently. This is derived from the SNR distribution generated by the simulation model. In Case 3 with $K = 3, 5, 7$, the performance and the power consumption are approximately only 1% lower than that of Case 2.

5.3.6 Summary

In this section, the DRP-based adaptive Viterbi decoder was described. Several Viterbi decoders with different constraint length K were implemented on the DRP-1 to evaluate impacts of the adaptive

Table 5.5: Throughput Comparison [Mb/s]

	T-R Separation		
	0.6km	2.0km	8.0km
Case 1	4.71	4.71	4.71
Case 2	8.89	6.48	4.95
Case 3	8.85	6.43	4.93
Case 4	8.89	6.48	4.95

Table 5.6: Power Comparison [mW]

	T-R Separation		
	0.6km	2.0km	8.0km
Case 1	1029.0	1029.0	1029.0
Case 2	493.6	718.8	974.1
Case 3	496.1	723.3	976.3
Case 4	(493.6)	(718.8)	(974.1)

computing on performance and power consumption. By switching the designs flexibly, the throughput varies from 4.71Mb/s to 9.95Mb/s, and power consumption from 429.0 mW to 1029.0 mW at the fixed throughput in response to the SNR. The power consumption can be saved up to 58.3% and throughput can be improved 2.1 times by switching designs appropriately if the distance between the base station and the mobile terminal is not so long.

The task-level time-multiplexed execution model is introduced in the adaptive Viterbi decoder in order to actually support several Viterbi decoders with various constraint lengths K. Evaluation result shows that the overhead caused by the configuration data transfers is not so large. This result is ensured by the fact that switching interval of the decoders is not so frequent with the actually-measured SNR distribution. However, the task-level time-multiplexed execution model results in additional power consumption and hardware cost of the configuration data memory with reducing the amount of context memories.

5.4 Data-resident and Configuration Moving: Multi-Core Structure

5.4.1 Motivation

In Chapter 4, area- and power-efficiency for a single DRP core has been investigated. The evaluation result shows that a relatively small-scale DRP core is almost preferable with respect to area and power. In order to improve the performance, it is required to exploit a task-level parallelism (TLP), and a multi-core structure of the dynamically reconfigurable processor cores should be investigated.

In conventional FPGAs, multi-task systems with partial reconfigurability have been proposed. These systems have a tiled structure, a single FPGA plane is divided into multiple Tiles with a certain size, and each Tile can be independently reconfigured. In these systems, however, each task communicates with each other via programmable routing resources of the FPGA. Partial reconfigurability for each Tile may be reduced for Tiles which are connected by the same path.

IMEC Marescaux *et al.* [96] have proposed their FPGA-based multi-task system in which a Network-on-Chip (NoC) is used for inter-task communications. The NoC can provide scalability and modularity to the system [97, 98] and simplifies the Tile-based partial reconfiguration. In Marescaux's NoC, a two-dimensional torus is introduced as a network topology. Each router for

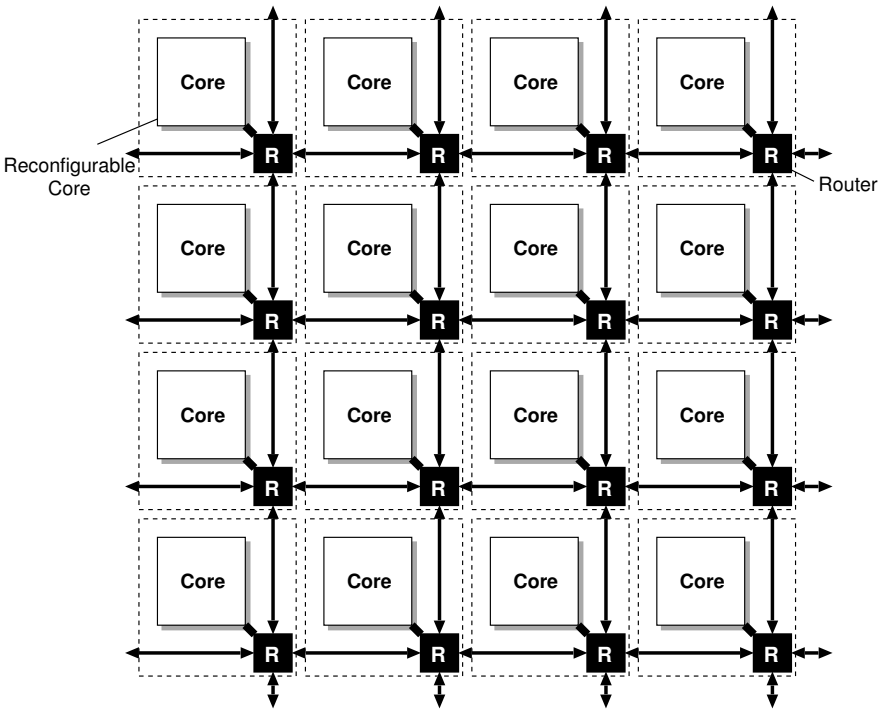


Fig. 5.11: Target Multi-Core Architecture

inter-task communication has two virtual channels, and provides wormhole packet switching and dimension-order routing (DOR) which is a deterministic routing algorithm. This system has been implemented on the Xilinx Virtex-II FPGA (XC2V6000).

In this work, we propose the multi-core dynamically reconfigurable processor in which each core performs task-level time-multiplexed execution [16]. Recent familiar chip-multiprocessors equip a simple on-chip bus for an inter-core network because of the relatively small number of cores. In the proposed architecture, however, a regular on-chip network is introduced in order to ensure each core scalability and modularity. We focus on a large application with a straightforward task flow such as JPEG codec, and evaluate the network performance with or without task-level time-multiplexed execution.

5.4.2 Target Multi-Core Architecture

Fig. 5.11 illustrates the target architecture model in this work. This architecture is supposed to have a homogeneous structure in which particular reconfigurable cores such as FPGAs or DRPs are regularly located. Actually, a heterogeneous structure containing embedded microprocessors, DSPs, and ASICs in addition to the reconfigurable cores is also acceptable. In this architecture, since each core is limited in a relatively small area, the dynamically reconfigurable processors are suitable because of their high flexibility and area-efficiency.

In this work, NEC Electronics' DRP is chosen as a basic core architecture, and each core is

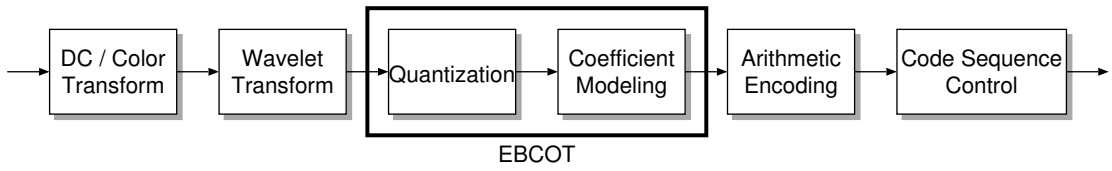


Fig. 5.12: Task Flow Graph of JPEG2000

connected to a single network router as shown in Fig. 5.11. For on-chip networks, packet transfer techniques for conventional multiprocessor systems are commonly used. Relatively simple topologies such as two-dimensional mesh, torus, and tree structure are popular. Yamada *et al.* [99] have proposed Fat H-Tree topology for reconfigurable systems. For routing algorithms, we can select an adaptive routing algorithm which dynamically determines the routing path depending on the network traffic jam and a deterministic routing algorithm in which the routing is statically specified. Since the adaptive routing can complicate the control flow and increase the hardware cost, it is not so used in cost-sensitive NoCs.

Also in this architecture, since each core is supposed to require small hardware cost and high area-efficiency, the small-scale network composing of simple network routers and interfaces is preferable. For this reason, each of the cores is connected by a two-dimensional mesh-based network, and it supports bidirectional inter-core communication. In addition, the dimension-order routing, which is the representative deterministic routing algorithm, is adopted for this architecture.

In contrast, in recent reconfigurable systems, a dedicated configuration bus which is separated from a system bus for data communications is commonly used for configuration data transfers. However, if the configuration data transfer does not frequently occur, the utilization of the dedicated configuration bus can be quite poor. Meanwhile, since the configuration data is generally larger than processing data, sharing the configuration bus and system bus may result in a crucial traffic jam. In the proposed architecture, both the configuration data and processing data are transferred via the same on-chip network. This can provide an efficient configuration data transfer scheme because the configuration data can be delivered in parallel fully exploiting the bandwidth of the on-chip network.

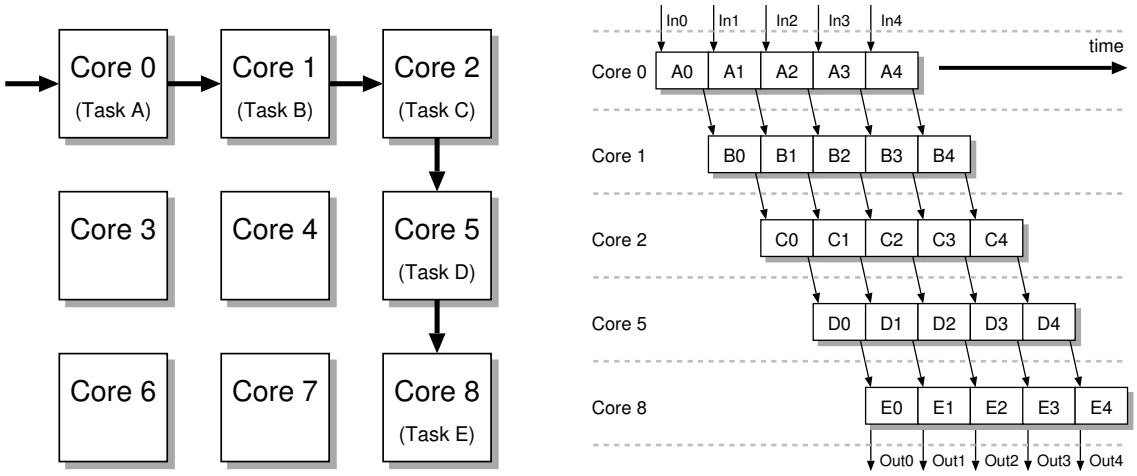
5.4.3 Application Execution Models

In the proposed architecture, the following two execution models can be supported.

(1) Stream-level Pipelining

Stream-based processings such as JPEG and MPEG have a series of computational tasks for a certain size of stream data arriving one after another. For example, the task flow of a JPEG2000 encoder is shown in Fig. 5.12.

In common NoCs, each of the computational tasks is allocated to a single core respectively, and the application can be executed in a pipelined manner. Fig. 5.13 illustrates an example of a task



(a) Task Mapping (Stream-level Pipelining)

(b) Data Flow (Stream-level Pipelining)

Fig. 5.13: Stream-level Pipelining

allocation with a stream-level pipelining and an inter-task data flow. In this example, the inter-task communication is restricted within adjacent cores. In the JPEG 2000 encoder, since an encoding task called EBCOT is the most computation-intensive task, it can be a bottleneck of the whole-system performance with only a single core. For the purpose of load-balancing of stream-based processing, EBCOT is divided into multiple subtasks such as Quantization and Coefficient Modeling and allocated to multiple cores.

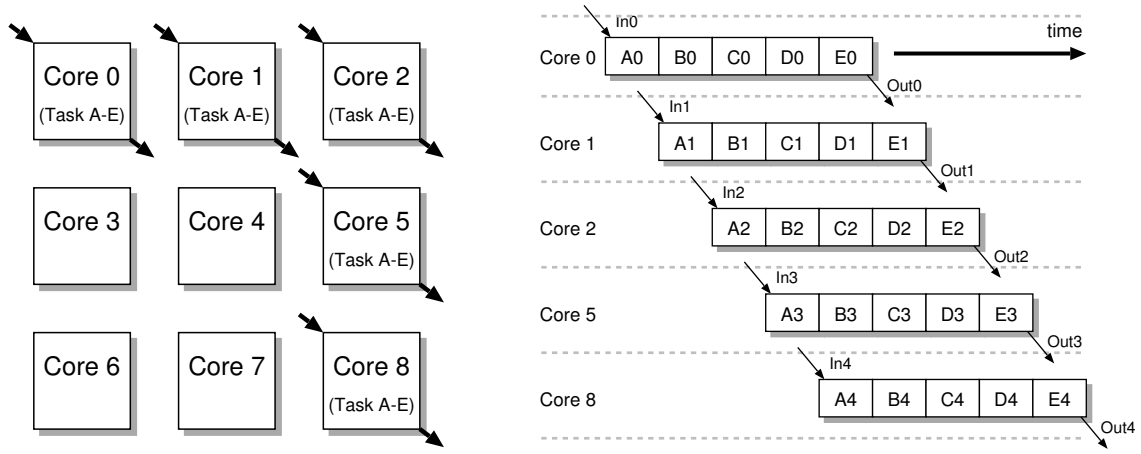
As shown Fig. 5.13(b), each core retains a particular task once it is allocated. The system can improve the throughput by executing computational tasks in the pipelined manner communicating to other cores. In contrast, since the task allocation for each core is statically fixed, configuration data transfers for switching tasks are not performed in this model.

(2) Configuration Moving

The stream-level pipelining is effective for data-parallel stream-based processing due to the TLP exploitation. However, in order to support more and more applications and multi-task scheduling cooperated with an embedded operating system, pipelining lacks in flexibility in terms of task scheduling.

For instance, similar to the JPEG2000 encoder shown in Fig. 5.12, an efficient transfer of streaming data among cores is significant for the stream-level pipelining. Thus, it is difficult to swap the task during the execution for each core, and the replacement of the whole application is rather practical. In contrast, for reconfigurable devices such as FPGAs and DRPs, a different execution model can be considered for the proposed architecture.

In this model, each core executes multiple tasks based on the task-level time-multiplexed execution model independently of the other cores. A task is executed on a particular core, and the



(a) Task Mapping (Config. Moving)

(b) Task and Data Flow (Configuration Moving)

Fig. 5.14: Configuration Moving

configuration data corresponding to the next task is transferred to the core, and after that, the task is deactivated and the next task is dispatched onto the same core. If the next stream data is present, the same tasks are dispatched onto other core and executed switching the tasks. In this method, once a certain stream data is transferred to a core, the data is resident on the core until the task flow of the application is finished. Therefore, inputs and outputs for each core ideally occur at only the beginning and end of the application. We call this execution model the *configuration moving* method, and an example of the task mapping and the data flow are shown in Fig. 5.14.

If each reconfigurable core has a sufficient context memory to hold all of the computational tasks, the configuration moving is not needed and the traffic data on the network can be minimized. In addition, if the target application has a strict data parallelism, more reconfigurable cores can improve the throughput due to the stream-level parallel processing. However, each task is individually designed and compiled to a single configuration data set, and it simplifies the task designs. Thus, the task switch requires quite a few configuration data transfer, and it can decrease the throughput because of the increase in network traffic caused by a large configuration data transfer. However, it can provide a more flexible task switching mechanism compared to the stream-level pipelining, and it is an indispensable feature for a multi-task system.

5.4.4 Simulation Environment

In this work, we evaluate the network performance of both the stream-level pipelining and the configuration moving method using traffic patterns of JPEG encoder which is one of the basic stream-based processings.

Fig. 5.15 shows the simulation environment of the proposed architecture. For simulating the on-chip network, we use a flit-level network simulator described in C++ language. The whole algorithm

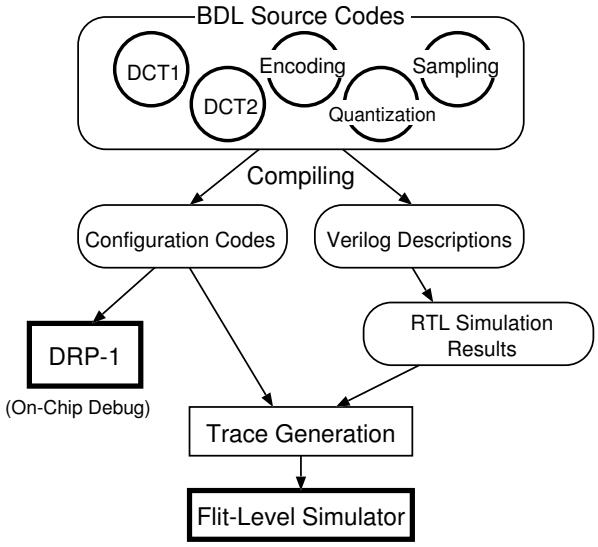


Fig. 5.15: Multi-Core Simulation Environment

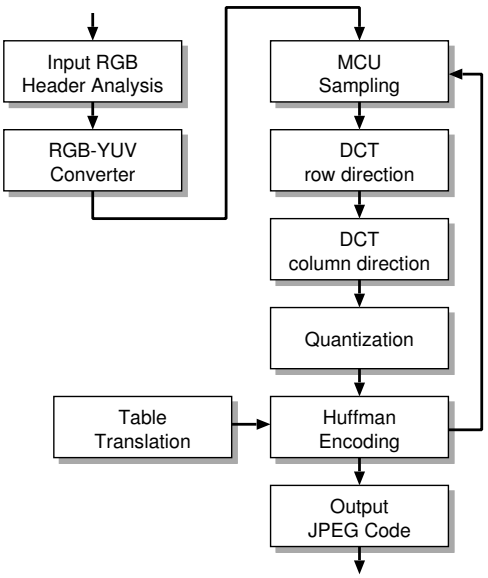


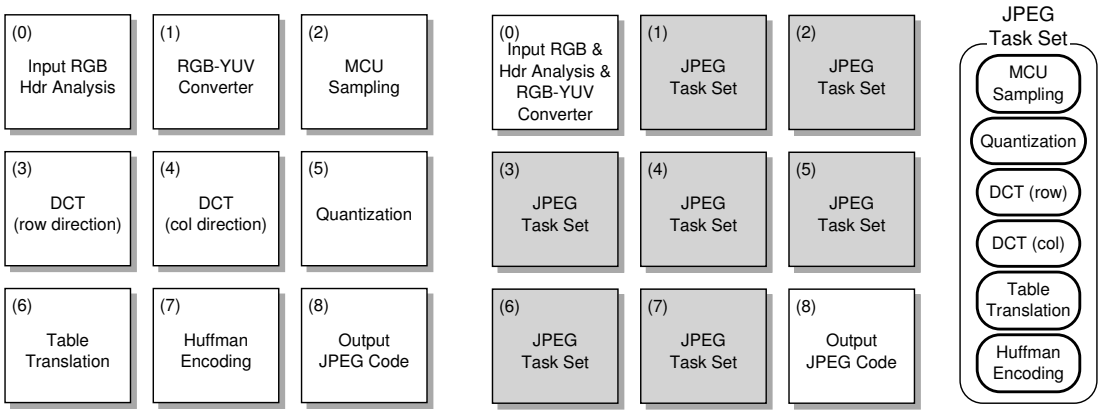
Fig. 5.16: Task Flow of JPEG Encoder

of the JPEG encoder has been described in C language, and we have traced the inter-task communication pattern from the software simulation. By scaling the time axis of the traced data, we can measure throughput and latency for various injecting data rates.

The basic structure of the on-chip network evaluated in this work is as follows. For the network topology and routing algorithm, two-dimensional mesh-based network with 3×3 size and dimension-order routing algorithm are chosen, respectively. Each network router has five IO ports, one is for the connection to the DRP core, and the others for connections to adjacent routers. In addition, a simplified model of an IO buffer and a crossbar switch with its controller are used for a switching mechanism of each router. The network requires three clock cycles to transfer a header flit to adjacent routers or the DRP core. The channel width of each router is 64-bit wide, and the data width of the IO ports of the DRP core is also 64-bit. For a packet switching method, the wormhole switching method is used, and a packet length is up to 16 flits including a header flit.

Since the core architecture is DRP-1 in this work, and each task must be executed on DRP-1 device, the JPEG encoder is divided into computational tasks as shown in Fig. 5.16. Each of the computational tasks is described in BDL and compiled by the DRP compiler. The configuration data corresponding to each task has been generated and verified on the real DRP-1 chip.

The task flow shown in Fig. 5.16 is mapped onto the proposed architecture. Fig. 5.17 illustrates mapping examples for the stream-level pipelining and the configuration moving, respectively. Since the JPEG encoder has a quite rectilinear task flow, it is feasible to easily design an efficient pipeline among computational tasks. Therefore, almost ideal mapping can be achieved for the stream-level pipelining, and the on-chip network barely has packet collisions (Fig. 5.17(a)). In contrast, the task combined with Input RGB and RGB-YUV Convert and the task for JPEG Code Output are fixed onto Core 0 and 8, respectively. The other tasks can be executed on the other seven cores by switching the



(a) Stream-level Pipelining (b) Configuration Moving

Fig. 5.17: Mapping Example of JPEG Encoder

configuration data in a data-parallel manner. In this thesis, it is assumed that only Core 0 can transfer the configuration data to each reconfigurable core.

In the configuration moving method, suppose that each core can hold all of the computational tasks in its context memory. Then, the streaming data on the on-chip network is only RGB image data from Core 0 to the other cores and JPEG-coded data from each core to Core 8. However, since the input and output are restricted to one core respectively, independent data distribution and collection for seven cores can result in concentrated traffic for Core 0 and 8, and also throughput degradation. Therefore, we have designed the computational tasks so that input and output stream data for each task can be distributed and collected to/from each core in a pipelined manner.

The traced data used in the flit-level simulator is generated based on the post-synthesis RTL simulation results of each task. The individual tasks are implemented on the DRP-1, and we can measure execution clock cycles and the amount of data transfer for each task. In addition, a generated configuration data size is also analyzed. A simple task switching mechanism is modeled, and we add the transfer patterns of the configuration data for the configuration moving method to the traced data.

5.4.5 Evaluation Results

For evaluating the proposed architecture, several traffic patterns are used for the simulations. In addition to the trace data of the stream-level pipelining (denoted by PIPE), the following trace data for the configuration moving are evaluated:

- A64: the case in which the task is transferred from Core 0 for every task switching,
- F64: the case in which each core retains the largest DCT tasks (row and col) in its context memory, and the other tasks are transferred from Core 0,

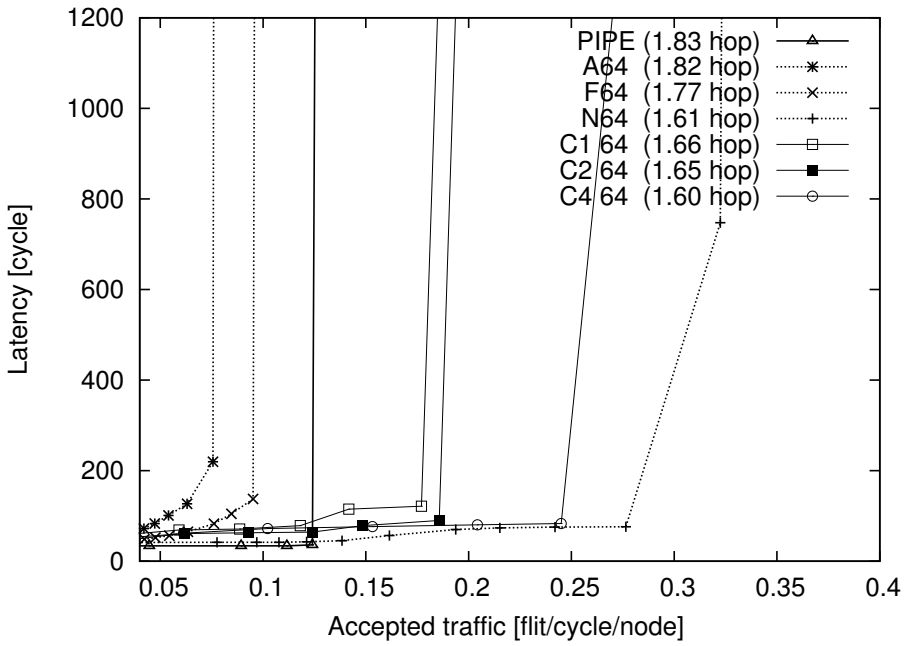


Fig. 5.18: Multi-Core Simulation Results

- N64: the case in which all of the tasks are assumed to be completely saved in the context memories for each core, and there is no configuration data transfer in the network, and
- C1, C2, C4: the cases in which each core has a configuration data cache, and the number of tasks to be cached simultaneously is assumed to be 1, 2, and 4, respectively, and the configuration data are shared among the cores.

Fig. 5.18 shows the evaluation results of the throughput (Accepted Traffic) and latency for each traffic pattern. As shown in this figure, N64 can provide the highest throughput compared to the other traffic patterns because the configuration and stream data transfers are minimized. PIPE can provide high throughput compared to A64 and F64. This is because the JPEG encoder is inherently suitable to the stream-level pipelining, and most stream packets can be efficiently transferred via the on-chip network without critical packet collisions. For A64 and F64, in contrast, the network traffic from Core 0 is concentrated at the configuration moving, and it results in a traffic jam on a particular path and consequently the decreasing throughput.

Compared to A64, it is possible for F64 to reduce the configuration data transfer, but it cannot lead to the significant throughput improvement. The main reason is that there still exist many traffic collisions because of the large amount of configuration data from Core 0, and the required time for task switching becomes a bottleneck of the on-chip network. In contrast, C1, C2, and C4 can distribute the transfer paths of the configuration data since each core can share the configuration data in the configuration data caches. Therefore, the throughputs of these traffic patterns outperform F64 and PIPE, and especially C4 can achieve the comparable performance to N64.

Table 5.7: Context Counts and Configuration Data Size for JPEG Encoder

Task Name	# of Contexts	Configuration Size [byte]
Header Analysis	2	2224
RGB-YUV Converter	5	21308
MCU Sampling	4	6292
DCT (row)	7	33504
DCT (column)	7	34544
Quantization	7	16604
Table Translation	4	8060
Huffman Encoding	10	18596

Table 5.7 shows the required number of contexts and configuration data size for each task. From this table, we can find that the configuration data is large even with the relatively small task. In addition, the configuration data for A64 seems to be huge compared to the amount of streaming data in pipelining method. For the required number of contexts, it has enough space to be stored in the DRP-1. In future versions of the DRP architecture, it is expected to be able to hold about 64 contexts in the on-chip context memory, and so we can believe that C2 and C4 are feasible solutions.

In this work, the DRP core with 8 Tiles is assumed. As shown in Chapter 4, a relatively small-scale DRP core with 2 or 4 Tiles is area-efficient for most applications. Assuming a smaller core in the proposed architecture, the configuration data transfer is expected to increase because more and more contexts are required to design each task. Although a more powerful network for the configuration data transfer can be expected to overcome this issue, it has a great advantage to expand available contexts for each core with respect to the throughput.

5.4.6 Summary

In this work, we proposed a multi-core dynamically reconfigurable processor architecture. Each reconfigurable core is connected by the on-chip network and supports the task-level time-multiplexed execution. In the proposed architecture, we investigated two application execution models: the stream-level pipelining and the configuration moving.

NEC Electronics' DRP-1 as a core architecture and a simple structure of the on-chip network were used. The computational tasks of JPEG encoder were implemented on the DRP-1, and we simulated the proposed architecture by using the flit-level network simulator and evaluated the throughput based on various traffic patterns.

From the evaluation result, the configuration moving can cause crucial degradation of throughput compared to the stream-level pipelining because of the large amount of configuration data. In contrast, by sharing the computational tasks in context caches among reconfigurable cores, we can improve the throughput due to the distribution of network paths. In addition, if each core can hold all

of the tasks in a largely expanded context memory, it can provide comparable throughput to the ideal case. In this case, since the configuration moving method can fully exploit data-level parallelism, more and more cores can achieve linear speedup and high scalability. With the increase of cores, the high throughput can be achieved by the high-bandwidth on-chip network.

Chapter 6

LSI Design of Time-multiplexed Programmable PE Array: MuCCRA-1

This chapter describes Multi-Core Configurable Reconfigurable Architecture (MuCCRA) and its first silicon implementation, MuCCRA-1. We develop a real chip for a dynamically reconfigurable processor, and quantitatively investigate the time-multiplexed execution models based on it.

6.1 MuCCRA Project Overview

The MuCCRA project aims to develop a methodology and a framework to design highly configurable dynamically reconfigurable processor arrays (DRPAs) for various target applications. As shown in Fig. 6.1, the final goal is to develop a multi-core SoC with an embedded microprocessor and multiple DRPAs as off-loading engines connected with NoC. Each of the DRPA cores is configurable or customizable for an application, while the data transfer through the NoC and context and task control mechanisms are common in all of the DRPA cores. Of course, the MuCCRA architecture supports the context-level and task-level time-multiplexed execution models investigated in Chapter 4 and Chapter 5, respectively, and the development of an efficient control scheme is one of the most important jobs in the project.

In this chapter, MuCCRA-1, the first prototype chip of this project, is described. The position and objective of the MuCCRA-1 in this thesis are as follows:

- to implement a real silicon chip of a dynamically reconfigurable processor which completely provides the context-level and task-level time-multiplexed execution models,
- to investigate speed and area/power trade-offs associated with time-multiplexed execution models based on the real chip, and
- to evaluate the hardware cost for the architectural support of the time-multiplexed execution models.

Throughout the MuCCRA-1 implementation and evaluation, we investigate the effectiveness of

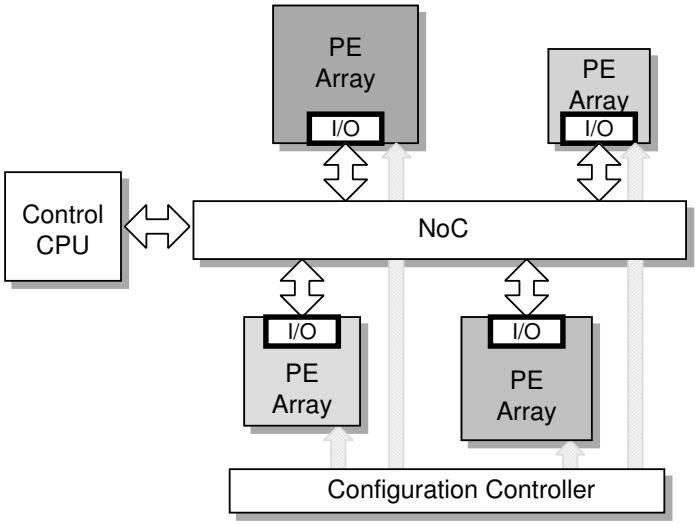


Fig. 6.1: MuCCRA Overview

the time-multiplexed execution models.

6.2 MuCCRA-1 Architecture

The MuCCRA-1 is the first prototype chip of the MuCCRA project, and it has been developed as a reference chip for evaluating performance, area, and power consumption. For this purpose, we have investigated the most popular PE-array structure which is adopted in recent DRPAs. On the other hand, we propose novel schemes for context and task control mechanism which can be used in all cores.

6.2.1 PE Array Architecture

The MuCCRA-1 is a coarse-grained dynamically reconfigurable processor which is based on a small-scale and homogeneous PE array. As shown in Fig. 6.2, the MuCCRA-1 is composed of a 4×4 24-bit PE array, four 24-bit dedicated multipliers (MULTs) and four 24-bit distributed memory modules (MEMs) at the left and bottom edge of the PE array, respectively. Each MULT multiplies two 24-bit data words, and lower 24-bit of the product is available as an output with a clock delay. Each MEM is a 24-bit $\times$ 256-word and 2-ported SRAM module, and all modules are double-buffered so that operations of the PE array and input/output can be performed independently. While one module is used by the PE array, the other can communicate with the outside via a 64-bit interface.

Like common FPGAs, an island-style interconnection network is used in the MuCCRA-1, and it is based on a two-dimensional array of PEs with routing channels located between the rows and columns. Input or output of each PE can connect to channels adjacent to it via a connection block. Data can be transferred along multiple routing channels in any direction, horizontally or vertically by using Switching Elements (SEs) at the channel intersections. The SE is a set of simple programmable

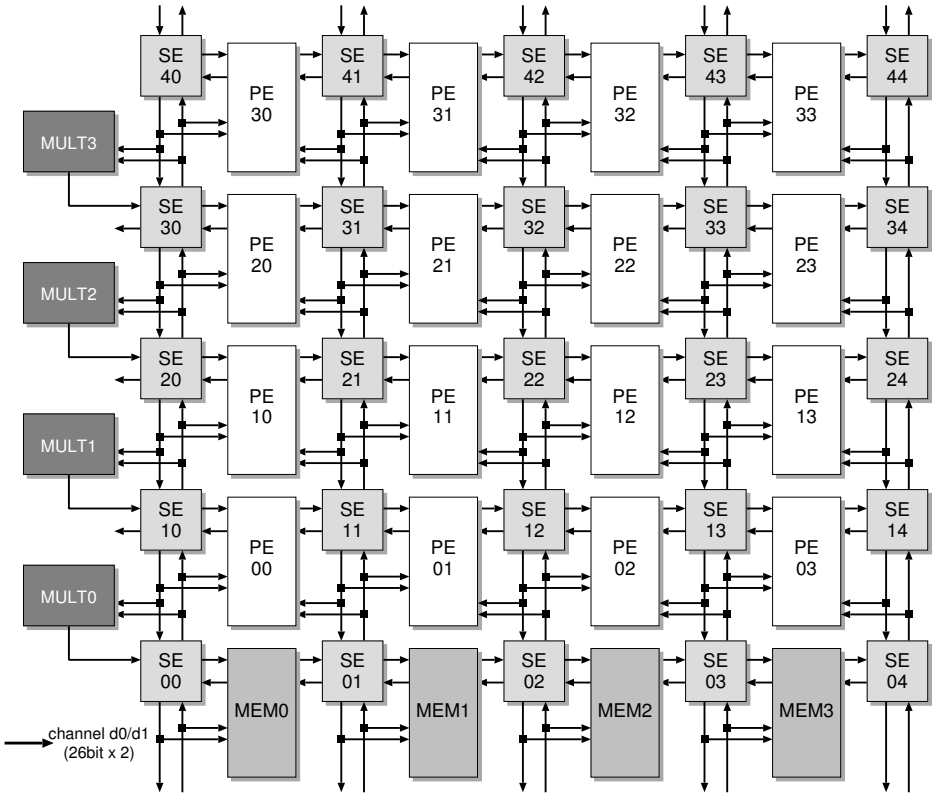


Fig. 6.2: PE Array Architecture

switches which can connect to adjacent SEs. The MuCCRA-1 has two 24-bit links called (d0, d1) from side to side and up and down. Although it is omitted in Fig. 6.2, writing data can be transferred into MEMs via the feedback wires from the uppermost SEs.

6.2.2 PE Architecture

Each PE of the MuCCRA-1 consists of a 24-bit PE Core, connection blocks for the connections to global routing resources, and a context memory.

The detailed architecture of the PE Core is illustrated in Fig. 6.3. Since one of the target applications of the MuCCRA-1 is image processing such as JPEG encoder for embedded systems, in order to treat RGB image data efficiently, the PE Core is designed as a simply structured 24-bit processor. Similar to common dynamically reconfigurable processors, it is composed of a Shift and Mask Unit (SMU), an Arithmetic and Logic Unit (ALU), and a register file (RFile). The SMU supports various types of shift & mask operations and supply of a constant value, and the ALU provides addition/subtraction, comparison, and logical operations. Both the SMU and ALU can perform half-word operations, in which 24-bit datum is treated as two 12-bit data, and are operated simultaneously. The RFile is a 24-bit wide and 8-entry deep register file, and it has a read/write port (A) and a read-only port (B). For avoiding combinatorial loops, an output of the ALU can be connected to an input of the SMU, but the opposite connection is not allowed. On the other hand, each RFile can connect with all

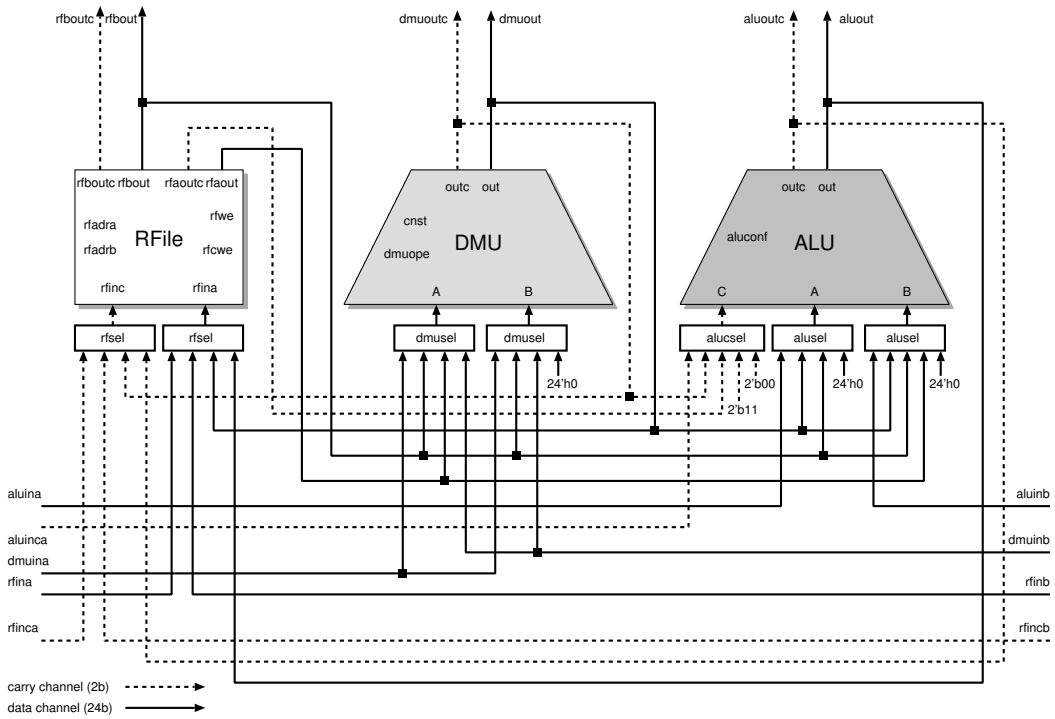


Fig. 6.3: PE Core Architecture

of inputs and outputs of the ALU and the SMU.

The context memory is a 64-bit × 64-entry memory, and it can hold the configuration data which contains operational instructions of the PE Core and connection instructions to global routing resources. The configuration data for one context of a PE is 64-bit wide, and each context memory can hold up to 64 contexts. The context control mechanism is described in Section 6.3.

6.2.3 SE Architecture

Each SE consists of four multiplexer-based programmable switches (SWs) and a context memory containing configuration data which specifies a destination of each SW. As shown in Fig. 6.4, the SW directs an entering data to a desired output direction for each link (d0, d1). Note that an input from the North is latched into a register in order to avoid combinatorial loops in the interconnection network, while inputs from other directions are unrestricted. The detailed structure of the inter-PE connection network of the MuCCRA-1 is described in the next subsection.

In general, a multiplexer-based switch consumes a larger area than a programmable switch of FPGAs in which a bidirectional data transfer is allowed by transfer gates. However, it is commonly used for island-style dynamically reconfigurable processor since short-term conflicts of outputs at dynamic reconfiguration time can be avoided.

The configuration data for a context of SE is 16-bit wide, and each context memory of SE can hold up to 64 contexts similar to PE.

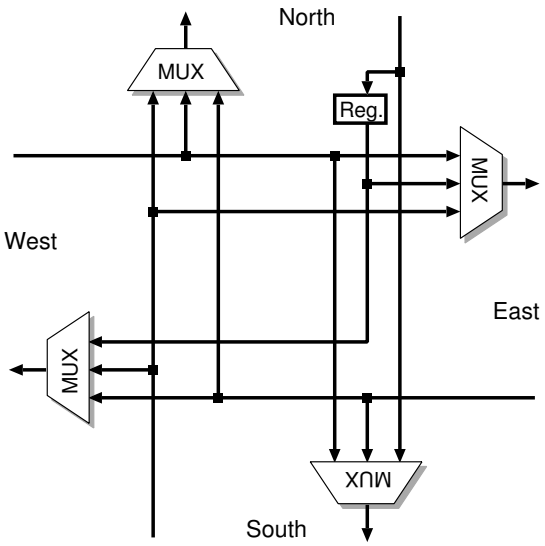


Fig. 6.4: Switch (SW) Architecture

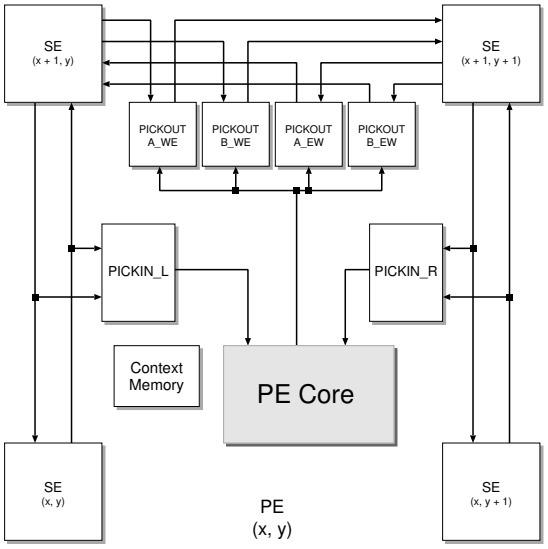


Fig. 6.5: Inter-PE Connections

6.2.4 Inter-PE Connection Network

The inter-PE connection network of the MuCCRA-1 is illustrated in Fig. 6.5. Each PE can select and take data from vertical global routing resources (d0, d1) in both sides of the PE via an internal input connection block called PICKIN. On the other hand, all outputs of ALU, SMU, and RFile of a PE can be transferred to horizontal links in any direction via the output connection block (PICKOUT).

For avoiding combinatorial loops, this network uses a same policy for deadlock avoidance in Turn model [100]. The connection turns from the down direction to either of the horizontal directions is only latched in an internal register of the destination SE, while other connections are allowed without any restrictions. Similarly, data from an upper PE to a lower PE must be stored in an RFile of the lower PE, while outputs of lower PEs are allowed to connect to inputs of upper PEs without any restrictions.

These restrictions are naturally fulfilled if the following typical datapath structure is mapped on to the PE array. The source data is read out from distributed memory modules (MEMs), processed upward with several PEs, and computation results are written into RFiles or MEMs via feedback lines from the topmost SEs.

6.3 Control Mechanisms of MuCCRA-1

There are several control mechanisms in the MuCCRA-1 architecture. For an architectural support of the context-level and task-level time-multiplexed execution models, the MuCCRA-1 equips a Context Switching Controller (CSC) and a Task Configuration Controller (TCC), respectively. The CSC is a simple program counter-based context controller with a context-branch control. The TCC is a task controller which has an on-chip configuration data memory, and it can perform a background

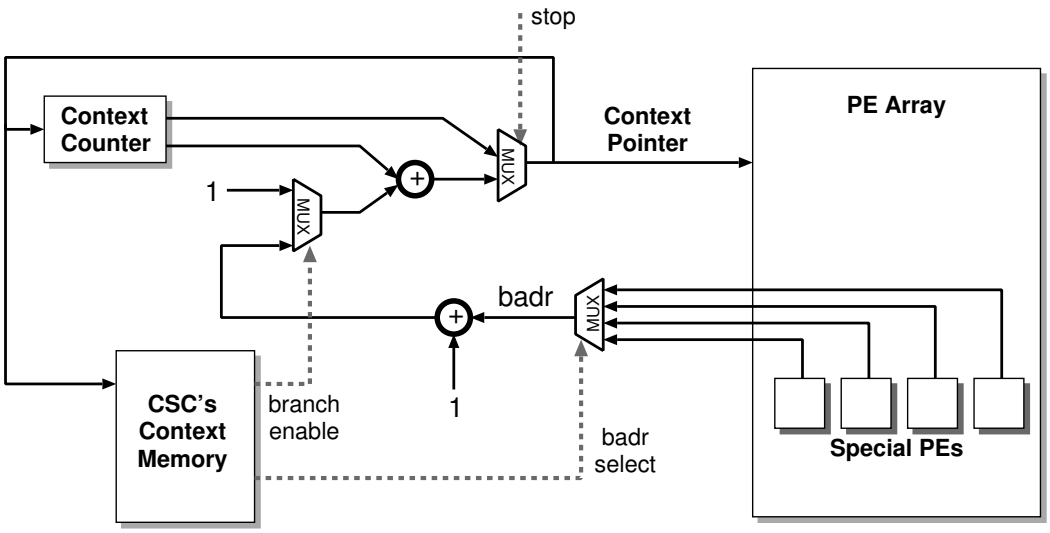


Fig. 6.6: Context Switching Controller (CSC)

configuration data transfer. In order to enhance the transfer speed, a multicast transfer scheme called RoMultiC is implemented in the TCC. Furthermore, similar to the context memory, the distributed data memory (MEMs) are also double-buffered and controlled by an independent I/O controller.

6.3.1 Context Control Mechanism

As mentioned in Section 3.1.2, several methods have been proposed for context switching control of multi-context dynamically reconfigurable devices. However, through the application designs on the DRP-1, we have found that a highly flexible control mechanism which requires the complicated hardware since most states transit to a single destination state. In the MuCCRA-1, we adopted a simple method using a context pointer similar to a program counter used in conventional computers. Each reconfigurable element of the MuCCRA-1 (PE, SE, MULT and MEM) provides a context memory which stores multiple sets of configuration data. In every clock cycle, each of the elements reads the configuration data from its context memory according to a context pointer delivered from the CSC. By changing the context pointer, the context is changed in a clock cycle. Note that the CSC itself is a reconfigurable element, and reads own configuration data from its context memory.

As shown in Fig. 6.6, the next context is decided with a context counter which holds the current context number. If there is no branch request, the context counter is simply incremented. For a context branch, a branch target context number ("badr") is transferred from an RFile of special PEs in the PE array. In the MuCCRA-1, four PEs at the edge of columns are special PEs which can generate a candidate of "badr", and final "badr" is selected with the configuration data of the CSC. By changing "badr" according to computation results, branches to multiple destinations like state transition-based control can be implemented in this mechanism. By using "-1" as "badr", the same context can be executed until the condition is satisfied.

6.3.2 Task Control Mechanism

The computational tasks for the MuCCRA-l are managed by the TCC. The TCC transfers a set of configuration data for a task from the configuration data memory to unused space of each context memory in all reconfigurable elements without disturbing the PE array as possible. The configuration data memory which is equipped by the TCC is an 88-bit $\times$ 512-entry on-chip SRAM module for storing all target tasks.

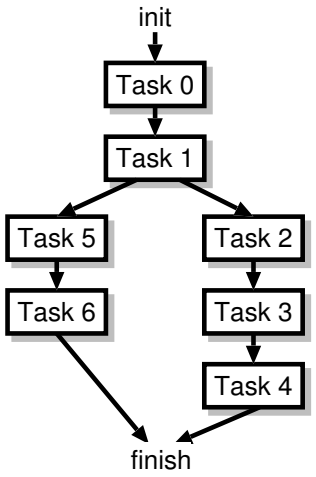
The number of available contexts of the MuCCRA-1 is 64, and it is larger than most existing DRPAs (16 of DRP-1, 4 of DAPDNA-2, and 32 of ADRES). However, since recent complicated jobs for multi-media processing require many contexts, they are difficult to be stored even with 64 contexts. Fortunately, most of the stream-based applications can be divided into independent computational tasks. Therefore, the MuCCRA-1 has a virtual hardware mechanism based on the task-level time-multiplexed execution although it is rarely implemented in a real chip.

The cryptographic accelerator and the adaptive Viterbi decoder have dynamic task transition controls including data-driven and event-driven controls as introduced in Section 5.2 and Section 5.3, respectively. However, they are not efficient to the applications which have a fixed task flow such as a JPEG encoder. The virtual context number proposed in [101] also requires complicated hardware. In the MuCCRA-1, the TCC supports a statically-defined task transition diagram for switching tasks. Although the TCC supports the static task transition, the dynamic task transition control can be also performed by collaborating with an embedded microprocessor.

The task transition diagram is described in a simple table called Task Flow Table (TFT). The TCC transfers configuration data for a task according to a TFT entry. An example of a TFT and its corresponding task flow graph are shown in Fig. 6.7. Each TFT entry for a task consists of the following items:

- start: a starting address of the configuration data memory which provides the configuration data for the target task,
- size: number of entries for the task in the configuration data memory,
- context size: number of contexts used in the target task,
- branch task: the next task number when a task branch is taken,
- default task: the next task number when a task branch is not taken, and
- jobend: a flag for the last task.

In the MuCCRA-1, while a task is executing on the PE array, the configuration data of “default task” are pre-loaded to the unused space of the double-buffered context memory without stopping the execution. In the example shown in Fig. 6.7(a), the configuration data for “default task” of Task 0, Task 1, is transferred during the execution of Task 0. Fig. 6.8(a) shows the context memory of each reconfigurable element when the configuration data transfer for Task 1 is finished. The sum of



(a) Task Flow Graph

	start	size	csize	branch	defalut	jobend
Task 0	0	60	20	---	1	---
Task 1	61	50	26	5	2	---
Task 2	112	80	48	---	3	---
Task 3	179	44	16	---	4	---
Task 4	224	60	36	---	---	1
Task 5	285	42	14	---	6	---
Task 6	328	36	8	---	---	1

(b) Task Flow Table

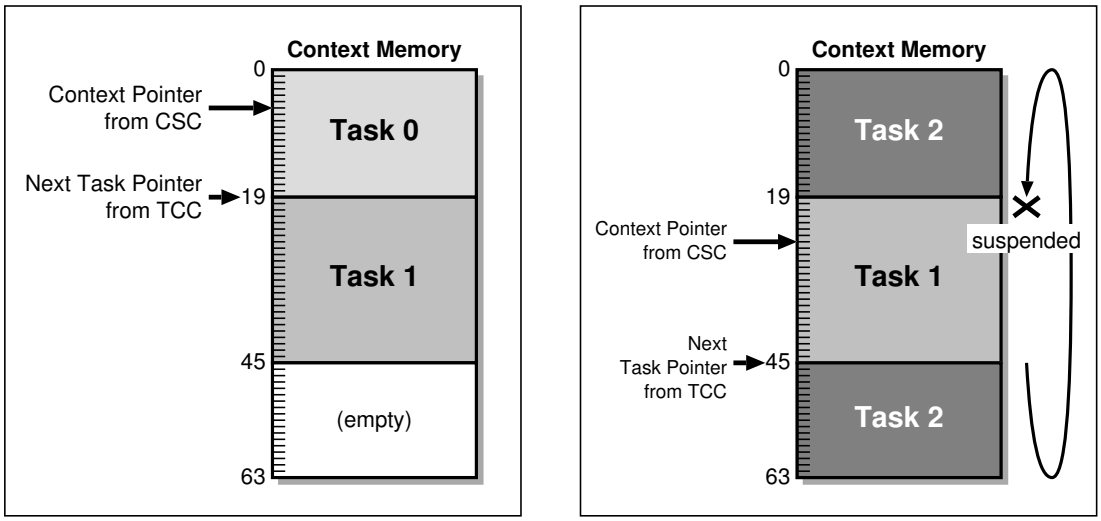
Fig. 6.7: Task Description of MuCCRA-1

context counts of Task 0 (20) and Task 1 (26) is 46, and so there is enough space for the next task to be transferred. However, in order to make the control simple, the successive tasks of Task 1 are not transferred during the execution of Task 0 even if the context memory has more empty space. After transferring the configuration data for the "default task" (Task 1), the operation of the TCC is suspended.

When the execution of Task 0 is finished, the TCC receives a task switching signal from the PE array. In this case, since the configuration data transfer for Task 1 has been finished, the PE array starts the execution of Task 1 immediately. At the same time, the TCC starts transferring the next default task Task 2. The context memory is used as a cyclic buffer, and the configuration data for Task 2 is stored during the execution of Task 1. In this case, however, since the sum of context counts required for Task 1 (26) and Task 2 (48) is larger than 64, the configuration data transfer stops when the context memory becomes full (Fig. 6.8(b)), and the TCC is suspended. In this example, Task 1 indicates a branch task (Task 5) which can be selected with a similar mechanism of a context branch. One of four special PEs can send a branch signal from its RFile to indicate the task branch. The used special PE is selected by the configuration data of the CSC.

When the execution of Task 1 is finished, if there is no branch signal from the special PE, the rest of the configuration data for the default task (Task 2) is transferred. Until all configuration data are transferred, the execution of the PE array is suspended. If a branch signal from the special PE is detected, Task 5 must be executed. In this case, the pre-loaded configuration data of Task 2 is overwritten with the configuration data for Task 5. During this transfer, the PE array is also suspended.

The virtual hardware can be implemented with this simple mechanism, since a context branch is



(a) Just after loading of Task 1

(b) During execution of Task 1

Fig. 6.8: An Example of Task Configuration

relative and the configuration data are relocatable. If each task is not too large compared with the size of context memory, most of the time for transferring the configuration data can be hidden except for the following cases:

- The required contexts for the current task and the default task are larger than the size of context memory 64.
- A task branch is taken.
- The current task is so small such that it is finished before transferring the default task.

In this method, each task is assumed to be implemented within 64 contexts, and a large task must be divided so as to keep the limitation.

6.3.3 RoMultiC: Multicasting Configuration Data

In order to enhance the speed of configuration data transfer from the configuration data memory to distributed context memory modules, a multicast mechanism called RoMultiC [102] is introduced in the MuCCRA-1.

In this mechanism, a single configuration bus is provided from the configuration data memory to the distributed context memory modules like other configuration delivering methods. However, instead of indicating the target context memory module by a serial address, multicast bits for rows and columns of the PE array are used as shown in Fig. 6.9. The configuration data on the bus is transferred to all elements whose row direction bit and column direction bit are both 1. The same configuration data is transferred to a rectangle region in the PE array specified with the bits. Since a

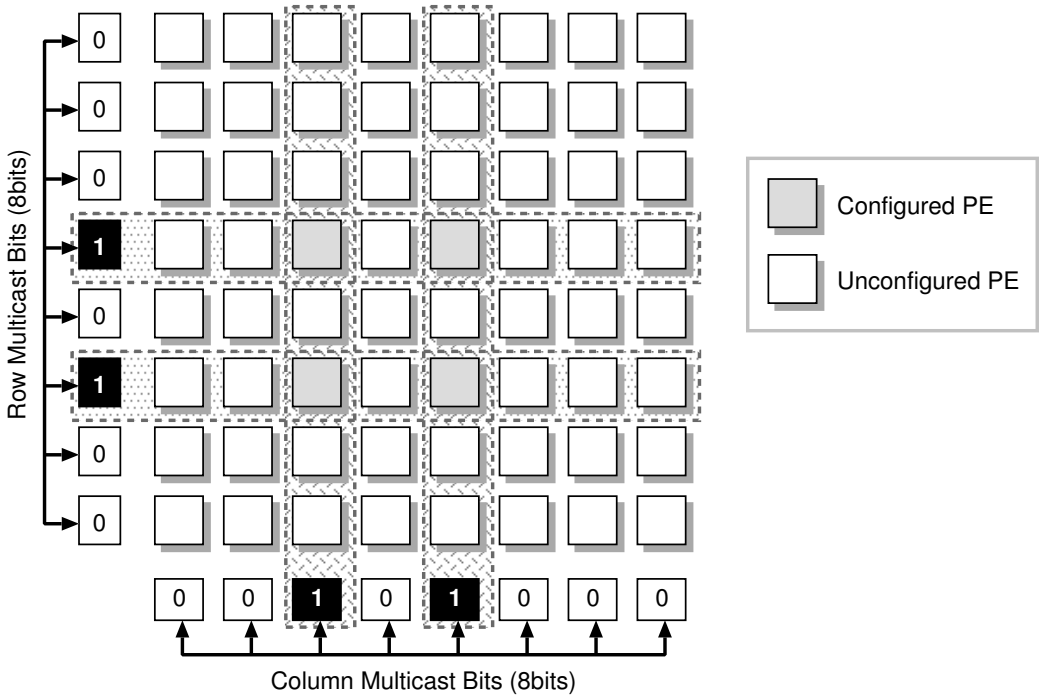


Fig. 6.9: RoMultiC (For 8×8 PE Array)

number of PEs and SEs use the same configuration data for SIMD-style parallel processing, transfer cycles are reduced by the RoMultiC very much. Also, since the multicasting data are shared with multiple PEs and SEs, the total requirement of the configuration data memory can be reduced.

6.3.4 Input/Output Control Mechanism

Input/Output data is transferred directly to/from MEMs independently of a computation in the PE array. Each MEM consists of two memory modules which work as a double-buffered data memory. Although a switch of the double buffer is synchronized with context switching, I/O data transfer is controlled with its own controller independent of the computation in the PE array.

As shown in Fig. 6.10, while the PE array computes using data stored in a buffer of MEM modules, previous results are sent out and the next data to be computed are received through the I/O controller. When both the computation in the PE array and the data transfer through the I/O controller are finished, the CSC switches the buffer to start the next computation and I/O data transfer. The I/O data transfer is overlapped with the computation in the PE array. In most stream-based processings, the time for I/O can be completely hidden [103].

6.4 Physical Implementation of MuCCRA-1

The MuCCRA-1 was fabricated onto a 5.18-mm square die in Rohm 0.18μm CMOS technology with 189 I/Os. The RTL model of the MuCCRA-1 was described in Verilog-HDL, and we used Synopsys

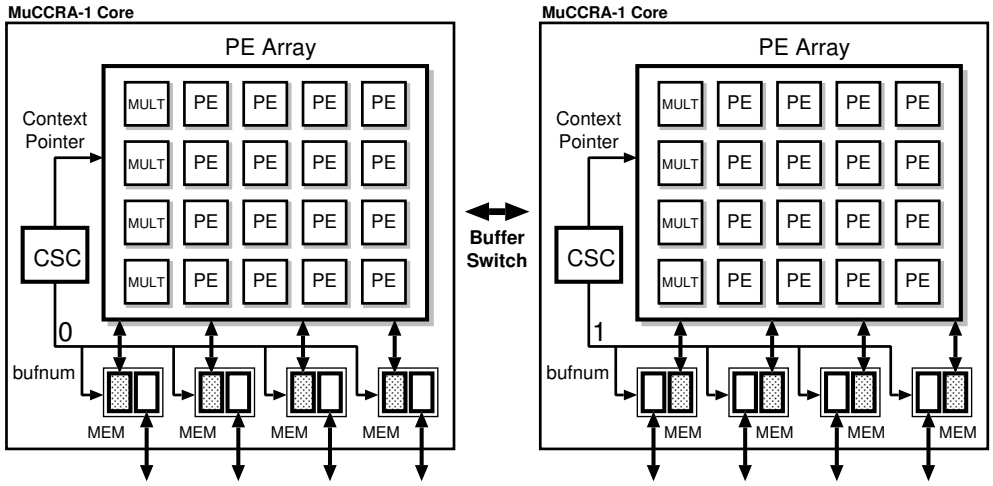


Fig. 6.10: I/O Interface of MuCCRA-1

Table 6.1: Cell Usage and Area of MuCCRA-1

Cell Counts	121,305
Gate Counts	1,125,231
Total PE-array Area	10.89 [mm^2]
Total Memory Area	7.33 [mm^2]

Design Compiler 2006.06-SP2 and Cadence SoC Encounter 5.2 for logic and physical syntheses. Fig. 6.11 shows the layout of the MuCCRA-1. The PE array is located as the same structure of the diagram shown in Fig. 6.2. The TCC with the configuration data memory and the CSC are located in the center of the chip.

Table 6.1 shows the usage of standard cells and the total area after the layout. The total area of memory modules including MEMs, context memory modules, and the configuration data memory occupies 67% of total core area. The context memory modules occupy 71% of total area of all memory modules (46% of the core area). The current implementation uses a relatively large context memory in order to store a large number of contexts. However, if the virtual hardware mechanism does not frequently stall with smaller number of contexts, the size should be reduced.

Fig. 6.12 shows the area breakdown of the total area of the PE array including each context memory. The area of "Controller" in the figure includes the CSC and the TCC with the configuration data memory. As shown in Fig. 6.12, the area of PEs is dominant, and the ratio of Controller is not so large. The context control and configuration data transfer schemes used in the MuCCRA-1 can be implemented with a small amount of hardware (about 1.3%). Since the area for MULTs is only 1.6% in this implementation, multiplications can be implemented as a function of each PE.

In addition, the ratio of the area for a context memory to the total PE area becomes 55.7%. This means that the ratio between PE logic and context memory which can hold 64 contexts is

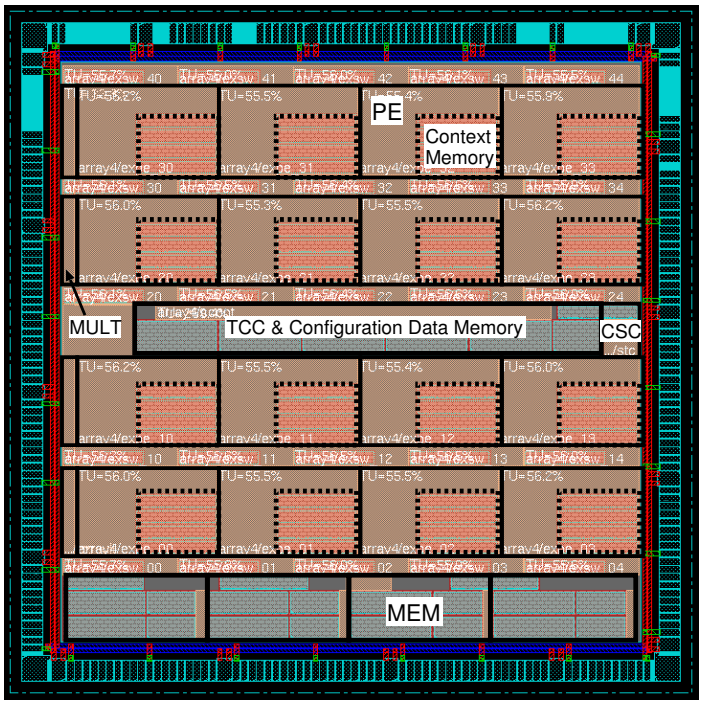


Fig. 6.11: Layout of MuCCRA-1 Chip

almost 1:1. The ratio of the context memory area required for storing configuration data to realize a hardware context of the PE logic becomes 0.019. The smaller the ratio is, the area-efficiency of time-multiplexing can be improved. The ratio in the MuCCRA-1 is smaller than that of IMEC ADRES (about 0.031) [68], and high area-efficiency can be expected.

6.5 Evaluation Using Some Applications

In this section, execution speed and power consumption of the MuCCRA-1 are evaluated with some multi-media programs based on the post-layout simulation.

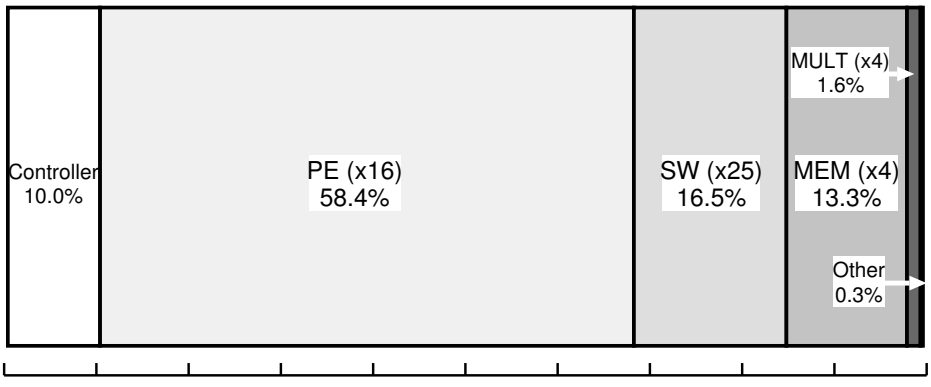


Fig. 6.12: Area Breakdown of each Reconfigurable Element

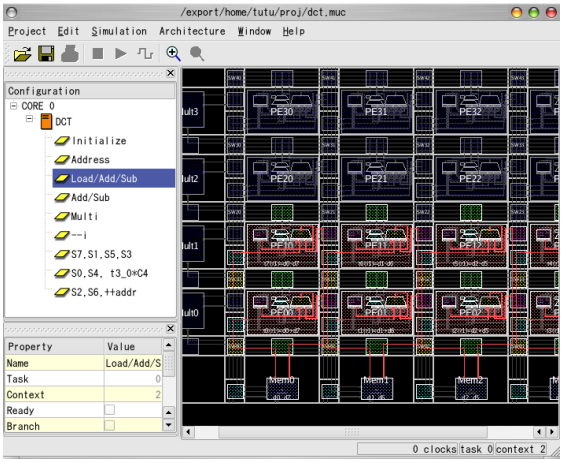


Fig. 6.13: MuCCRA-editor

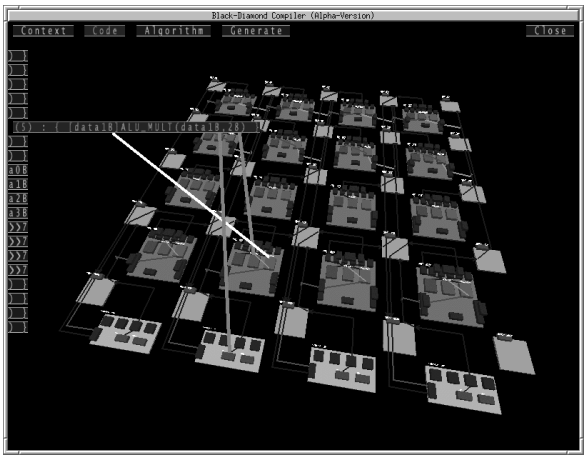


Fig. 6.14: GUI of the Black Diamond

6.5.1 Application Design Environment

The performance of DRPAs highly depends on the development tool. For the MuCCRA-1, two programming tools are available; one is a schematic editor called MuCCRA-editor, and the other is a retargetable compiler called Black-Diamond. In the MuCCRA-editor, a programmer defines configuration data by selecting operations of PEs and interconnections of SEs with GUI shown in Fig. 6.13. After selecting every module in a context, the configuration data which can be used in the post-layout simulation is generated. Although the place-and-route of a data flow graph into the PE array must be operated by the programmer, the order of the multicasting using RoMultiC is scheduled by the editor so as to minimize the time for configuration data transfer [104].

The Black-Diamond is a retargetable compiler which generates configuration data from a C-like algorithm description. The context scheduling, mapping, and place-and-route are automatically done by the compiler. The programmer can check the results by using the GUI shown in Fig. 6.14, and optimize them by inserting pragmas in the front-end description.

In this evaluation, four application programs; Discrete Cosine Transform (DCT) used in JPEG encoder, a simple image processing α-Blender, SHA-1 and a Viterbi decoder were implemented on the MuCCRA-1. Since the Black-Diamond compiler became available most recently, only α-Blender was designed using it. Other applications were developed by using the MuCCRA-editor.

6.5.2 Resource Usage

Table 6.2 shows the required resource including the number of required contexts (Context), total usage of PE units (ALU, SMU, and RFile), multiplier (MULT), and memory module (MEM). The values in low columns for each application show the ratio of the usage to all of available resources in consumed contexts. From the result, the DCT design consumes many MEMs and MULTs compared to the other designs, while it does not exploit PE units such as ALU so much. This is because the DCT

Table 6.2: Evaluation Results of Resource Usage

	Context	ALU	SMU	RFile	MULT	MEM
DCT	41	76 11.6%	217 33.1%	180 27.4%	56 34.2%	92 56.1%
α-Blender	8	14 10.9%	26 20.3%	41 32.0%	6 19.0%	12 38.0%
SHA-1	12	70 36.4%	128 66.7%	134 69.8%	0 0.0%	23 47.9%
Viterbi	13	125 60.1%	119 57.2%	130 62.6%	0 0.0%	42 80.8%

design reads an 8×8 RGB image data from the MEMs in parallel, and then executes the multiply-addition/subtraction using the MULTs. Although the α-Blender design also operates multiplications for a RGM data, the resource usage is not so large because of its small design scale.

In contrast, SHA-1 and Viterbi include no multiplications. Since the shift and logical operations are frequently operated in these applications, the usage of ALUs and SMUs is relatively high. And also, they consume a number of storage elements such as RFiles and MEMs because they have many table references and operations with constant values in RFiles. In addition, the data width of the SHA-1 design is 32-bit wide, and it is insufficient to implement a 32-bit datapath by using 24-bit elements of the MuCCRA-1.

6.5.3 Performance

Table 6.3 shows performance of each application executed on the MuCCRA-1 and the comparison to TI's TMS320C6713 DSP. The Clocks means the operational clock cycles for each application, and so they do not include the cycles for configuration data transfers and I/O transfers.

On the MuCCRA-1, delay time is dependent on the complexity of the design, and it becomes large if a context of the application has complicated long paths which go through many PEs and SEs. The evaluation result shows that delay time is from 24ns to 50ns, and the MuCCRA-1 works at relatively low clock rate. The MuCCRA-1 is twice faster than the DSP on α-Blender, while it is almost the same on DCT and SHA-1. Since the α-Blender design is originally small-scale and fully exploits the pixel-level parallelism, it can achieve a performance advantage compared to the DSP.

In contrast, the DCT design is too large application to be efficiently mapped onto the PE array of the MuCCRA-1. Since the DCT design is divided into more contexts by the step division, it has not obtained the performance improvement for its original parallelism. Moreover, in the case of SHA-1, since a 32-bit datapath has to be designed with 24-bit PEs, it results in enlarging the datapath and delay time. Consequently, the DCT and SHA-1 could not exploit the PE array so much for their parallelisms.

Table 6.3: Evaluation Results of Execution Time

	Architecture	Block Size [bit]	Clocks	Delay [ns]	Exec. Time [μs]
DCT	MuCCRA-1	1024	195	40.0	7.8
	DSP		1347	4.4	8.6
α-Blender	MuCCRA-1	8192	644	24.0	15.5
	DSP		8275	4.4	36.8
SHA-1	MuCCRA-1	512	418	50.0	20.9
	DSP		5276	4.4	23.5
Viterbi	MuCCRA-1	16	600	42.0	25.2

Table 6.4: Evaluation Results of Power Consumption

	PE	SE	MULT	MEM	Control	Other	Total[mW]
DCT	69.1	12.5	3.6	3.2	1.8	9.8	85.1
α-Blender	65.0	11.8	4.1	4.0	2.3	12.8	103.3
SHA-1	62.1	14.8	3.2	3.6	2.6	13.7	50.6
Viterbi	59.3	14.0	1.6	3.9	3.4	17.8	45.9

6.5.4 Power Consumption

The power consumption for each application has been measured from the post-layout simulation and the path-based power analysis by Synopsys PowerCompiler. Table 6.4 shows a power breakdown and the total power for each application with its maximum frequency.

The evaluation results show that power consumption is extremely low, 103.3mW at maximum, although almost no power saving techniques such as clock gating and operand isolation were applied. The table also shows that at least 59.3% of the design's power is consumed in the PEs. In contrast, the power consumption in controllers (Control) including the CSC and the TCC is small (1.8-3.4%) similar to their area. It also shows that the inter-PE connection network (SE) requires relatively large power. By introducing traditional power saving techniques, we can further reduce them. Moreover, we can find that the ratio of "Other" is so large (up to 17.8%). This includes the power consumption for buffers in clock trees and inter-PE connection networks. This result suggests that the power consuming in the interconnection network is also non-negligible.

6.5.5 Clock Cycles for Configuration Data Transfer

Table 6.5 shows clock cycles that are wasted due to the transfer of the configuration data for each application. The term "w/o RoMultiC" means the clock cycles used for transferring configuration data to all activated PEs and SEs one by one, and "Reduction" shows the ratio of reduced cycles with RoMultiC. The MuCCRA-1 takes 25 clock cycles for transferring a single context without RoMultiC.

Table 6.5: Clock Cycles for Configuration Data Transfer

	with RoMultiC	w/o RoMultiC	Reduction%
DCT	492	1025	52.0%
α-Blender	67	200	66.5%
SHA-1	220	300	26.7%
Viterbi	190	325	41.5%

By using RoMultiC, the number of clock cycles for transferring the configuration data is much reduced. In SHA-1, the reduction ratio is small since the number of PEs which execute the same operation is small. Compared with the clock cycles needed for the execution of the application, the clock cycles of the DCT design, which uses many contexts (41) are larger than that for execution. This means that the time for the configuration data transfer cannot be overlapped by the execution, and the execution is possibly stalled to wait for the next task being ready. In other applications, the configuration data transfer cycles are smaller than that for the execution, and so they can be hidden.

6.6 Summary

In this chapter, the MuCCRA-1, which is the prototype chip of the MuCCRA project, is described. The MuCCRA-1 supports both the context- and task-level time-multiplexed execution models completely. The MuCCRA-1 has been fabricated in Rohm 0.18μm CMOS technology, and a 4×4 24-bit PE array, four dedicated multipliers, and four distributed memory modules are integrated onto a 5.18mm-squared die. Each PE has a 64-depth context memory, and the context switching is controlled by a simple counter-based control mechanism. Moreover, the MuCCRA-1 provides a simple task controller which supports the background configuration data transfer scheme.

The evaluation result demonstrates that the MuCCRA-1 is up to 2 times faster than the TI's DSP operating at 225MHz with extremely low power consumption. And, the area overhead required for the control mechanisms of the time-multiplexed execution models is only about 1% of the total area. In contrast, the area of a context memory in each PE amounts to 55.7% of the PE area.

Each of the reconfigurable elements of the MuCCRA-1 has a 64-entry context memory. If the application can fully exploit the 64 contexts, the MuCCRA-1 has a great advantage of area- and power-efficiency. However, all of the context memory modules occupy as large as 46% of the total area. Since the MuCCRA-1 has only 16 PEs because of a chip size constraint, it must equip a large context memory in each element for a practical application. Moreover, considering an architectural support of the background configuration data transfer, 64-depth context memory is reasonable for the MuCCRA-1. Actually, the size of the context memory should be determined to suit the target application. Thus, the designers have to explore performance and area/power trade-offs for the PE-array size as remarked in Chapter 4.

Chapter 7

Conclusion

7.1 Thesis Summary

This thesis described two time-multiplexed execution models for multi-context dynamically reconfigurable processors. We investigated performance and area/power trade-offs in the time-multiplexed execution models and suggested a guideline to design an area- and power efficient architecture. Finally, we described MuCCRA-1, which supports the models.

(1) Context-level Time-multiplexed Execution Model

At first, in Chapter 4, a context-level time-multiplexed execution model was investigated by using NEC Electronics' DRP-1. In this model, each reconfigurable element has its own context memory for storing configuration data called context, and operational and connection instructions can be dynamically changed by switching the context in parallel. We modeled the context-level time-multiplexed execution model based on the DRP-1 and evaluated performance and area/power trade-offs while scaling the PE-array size. We proposed a parallelism-based method for estimating the optimum PE-array size in terms of area- and power-efficiency. Moreover, several applications were implemented on the DRP-1 with various PE-array sizes, and we compared the results in detail.

From the evaluation results, the optimum PE-array size in area- and power-efficiency can be correctly estimated. In addition, the result showed that a relatively small-scale PE array such as 64 or 128 PEs is in almost all cases efficient. However, there were cases where it was difficult to accurately estimate the speed and cost. This is mainly because the estimation model does not take the other characteristics of applications except for the parallelism into account.

(2) Task-level Time-multiplexed Execution Model

Secondly, Chapter 5 described a task-level time-multiplexed execution model targeting NEC Electronics' DRP-1, similar to the context-level time-multiplexed execution model. In this model, a configuration data memory for many computational tasks is equipped outside a PE array, and the

task can be dynamically switched by transferring the configuration data on demand. Each task is designed under a constraint of context memory capacity, and this method can be applied to a large application which exceeds the capacity. In this work, several applications were implemented and evaluated.

A DRP-based cryptographic accelerator for IPsec was developed. It supports multiple cryptographic tasks and switches them on demand. Each cryptographic task can achieve up to 7.8-times throughput compared to a MIPS64-compatible embedded microprocessor. However, run-time task switching causes a time overhead; an evaluation resulted in 20.1% of the total execution time. To address this issue, a background configuration data transfer scheme was incorporated resulting in an 80.7% reduction of the overhead.

Next, we developed an adaptive Viterbi decoder based on the task-level time-multiplexed execution model of the DRP-1. It can dynamically control performance and power trade-offs by switching the Viterbi decoders with different performance and power consumption depending on SNR. The evaluation result shows that the adaptive Viterbi decoder can reduce up to 58.7% of the power consumption. However, since the DRP-1 has an on-chip configuration data memory for storing many tasks, the power overhead of task switching cannot be evaluated.

We also investigated a multi-core architecture which consists of multiple dynamically reconfigurable processor cores connected by a NoC. Each of the cores executes computational tasks in the task-level time-multiplexed execution manner. By exploiting the data-level parallelism, the proposed architecture can reduce the inter-core communication and improve the performance. However, if the context memory capacity is insufficient, the configuration data must be moved between the cores, and the network performance can be degraded.

(3) MuCCRA-1 Physical Implementation

Finally, Chapter 6 explored the MuCCRA-1, which supports both the context- and task-level time-multiplexed execution models completely. The MuCCRA-1 has been fabricated in Rohm 0.18μm CMOS technology, and a 4 $\times$ 4 24-bit PE array, four dedicated multipliers, and four distributed memory modules are integrated onto a 5.18mm-squared die. Each PE has a 64-depth context memory, and the context switching is controlled by a simple counter-based control mechanism. Moreover, the MuCCRA-1 provides a simple task controller which supports the background configuration data transfer scheme.

The evaluation result demonstrates that the MuCCRA-1 is up to 2 times faster than the TI's DSP operating at 225MHz with extremely low power consumption. And, the area overhead required for the control mechanisms of the time-multiplexed execution models is only about 1.3% of the total area. In contrast, the area of a context memory in each PE amounts to 55.7% of the PE area.

7.2 Suggestions for Future Research

(1) Design Issues for Configurable Architectures

As described in Chapter 3, dynamically reconfigurable processors have a very wide design space including PE structures, intra/inter-PE connection networks, number of available multipliers and distributed memory modules, dynamic reconfiguration methods, context memory capacity, PE-array sizes, and so on. The datapath on the PE array is actually user-programmable, but the core architecture which is integrated to a system should be designed to suit target applications. Hence, a well-defined methodology and a framework to design a configurable architecture are important. We believe that such methodology and framework help for SoC designers to easily explore the design space and design an area- and power-efficient architecture for target applications. A few analytical researches derived from this kind of motivations have emerged like WPPA [57], and more flexible and practical experiments are also under investigation in our MuCCRA project [18].

(2) Future Researches for Interconnection Networks

In this thesis, none of the works for the interconnection networks between PEs or cores have been described. The advancing semiconductor technology leads to the issues of wire delay and power. Thus, innovative technologies are required at both architecture-level and circuit-level. From this motivation, recent architectures have a segmented or tiled structure in order to shrink long paths by a hierarchical network [64, 105]. NoC-based approaches based on packet transferring are also architectural solutions for low-latency and high-bandwidth interconnections [96]. However, the interconnection networks should be also configurable for each target application. The detailed trade-off analyses of network topologies and routing algorithms are required for high area- and power-efficient interconnection architectures.

One of the circuit-level solutions to the wire-delay issue is a three-dimensional IC technology. It stacks multiple dice using vertical interconnects such as through-wafer via and inductor-based wireless connections [106]. By holding several chips vertically, each chip area and the longest wire length in each chip can be linearly reduced. The stacked dynamically reconfigurable processor is an innovative approach toward the issue of stretched programmable wires. Currently, MuCCRA-Cube [107], which is a stacked dynamically reconfigurable processor based on inductor-based wireless connections, is now being investigating.

(3) Design Issues for Power-Aware Architectures

The power consumption is a non-negligible factor when considering the use of the dynamically reconfigurable processor in embedded products. We can say that they are still power-inefficient compared to ASICs. In addition, the dynamically reconfigurable processors require additional consuming power caused by the dynamic reconfigurability. Although several power analyses have been

published for power-efficient FPGAs, there is little literature which addresses the power-inefficiency problem of the dynamically reconfigurable processors. We can claim that a power analysis should be much closely performed.

For a low-power architecture, architectural techniques are required in addition to conventional low-power design techniques including a clock gating and an operand isolation. Furthermore, as a circuit-level low-power technology, we can suggest that context-level or PE-level supply voltage scaling [108], configurable dual-Vdd [109], and power gating [110] can be applied to the dynamically reconfigurable processors. Since the dynamic reconfigurability sometimes produces unused PEs or inactive contexts, these low-power techniques have a great advantage of reducing the dynamic and static power consumption for the unused or inactive parts.

Abbreviations and Acronyms

AES Advanced Encryption Standard
ALU Arithmetic Logic Unit
API Application Programming Interface
ASIC Application-Specific Integrated Circuit
BDL Behavioral Design Language
BER Bit Error Rate
CAD Computer-Aided Design
CMOS Complementary Metal Oxide Semiconductor
CPLD Complex Programmable Logic Device
CSC Context Switching Controller
DCT Discrete Cosine Transform
DMA Direct Memory Access
DRP Dynamically Reconfigurable Processor
DSP Digital Signal Processor, Digital Signal Processing
EDA Electronic Design Automation
FF Flip-Flop
FFT Fast Fourier Transform
FIFO First In First Out
FIR Finite Impulse Response
FPGA Field-Programmable Gate Array
HDL Hardware Description Language
ILP Instruction-Level Parallelism
IMDCT Inverse Modified Discrete Cosine Transform
IP Intellectual Property, Internet Protocol
IPsec IP Security
ISA Instruction Set Architecture
LIFO Last In First Out
LLP Loop-Level Parallelism
LUT Look-up-Table
MAC Multiplication and Accumulation

MIMD Multiple Instruction Multiple Data

MuCCRA Multi-Core Configurable Reconfigurable Architecture

NoC Network-on-Chip

PC Program Counter, Personal Computer

PCI Peripheral Component Interconnect

PE Processing Element

QoS Quality of Service

RFU Reconfigurable Functional Unit

RISC Reduced Instruction Set Computer

RTL Register Transfer Level

SE Switching Element

SHA Secure Hash Algorithm

SIMD Single Instruction Multiple Data

SNR Signal-to-Noise Ratio

SoC System-on-Chip

SRAM Synchronous Random Access Memory

TCC Task Configuration Controller

TFT Task Flow Table

TI Texas Instruments

TLP Task-Level Parallelism

TSMC Taiwan Semiconductor Manufacturing Company

VDEC VLSI Design and Education Center

VLIW Very Long Instruction Word

VLSI Very Large Scale Integration

Bibliography

[1] S. Brown, R. Francis, J. Rose, and Z. Vranesic. *Field-Programmable Gate Arrays*. Kluwer Academic Publishers, 1992.

[2] V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.

[3] F. Li, Y. Lin, L. He, D. Chen, and J. Cong. Power Modeling and Characteristics of Field Programmable Gate Arrays. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 24(11):1712–1724, November 2005.

[4] P. Jamieson and J. Rose. Enhancing the Area-Efficiency of FPGAs with Hard Circuits Using Shadow Clusters. In *Proc. of the 5th IEEE Int'l Conf. on Field Programmable Technology (ICFPT2006)*, pages 1–8, December 2006.

[5] G. Smit, P. Havinga, L. Smit, and P. Heysters. Dynamic Reconfiguration in Mobile Systems. In *Proc. of the 12th Int'l Conf. on Field-Programmable Logic and Application (FPL2002)*, pages 162–170, August 2002.

[6] A. Alsolaim, J. Becker, M. Glesner, and J. Starzyk. Architecture and Application of a Dynamically Reconfigurable Hardware Array for Future Mobile Communication Systems. In *Proc. of the 8th IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM2000)*, pages 205–214, April 2000.

[7] H. Amano. A Survey on Dynamically Reconfigurable Processors. *IEICE Trans. on Communications*, E89-B(12):3179–3187, December 2006.

[8] T. Sueyoshi and H. Amano, editors. *Reconfigurable Systems*. Ohmsha, 2005. (In Japanese).

[9] Y. Hasegawa, S. Abe, S. Kurotaki, V. M. Tuan, and H. Amano. Evaluation of Time-multiplexed Execution on the Dynamically Reconfigurable Processor. *IPSJ Trans. on Advanced Computing Systems (ACS)*, 47(SIG12(ACS15)):171–181, September 2006. (In Japanese).

[10] Y. Hasegawa, S. Abe, S. Kurotaki, V. M. Tuan, and H. Amano. Evaluation of Time-multiplexed Execution on the Dynamically Reconfigurable Processor. In *Proc. of the 4th Symp. on Advanced Computing Systems and Infrastructures (SACSIS2006)*, pages 135–142, May 2006. (In Japanese).

[11] Y. Hasegawa, S. Abe, S. Kurotaki, V. M. Tuan, N. Katsura, T. Nakamura, T. Nishimura, and H. Amano. Performance and Power Analysis of Time-multiplexed Execution on Dynamically Reconfigurable Processor. In *Proc. of the 20th IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS2006)*, April 2006.

[12] Y. Hasegawa, S. Abe, H. Matsutani, K. Anjo, T. Awashima, and H. Amano. An Adaptive Cryptographic Accelerator for IPsec on the Dynamically Reconfigurable Processor. *IEICE Trans. on Information & Systems*, J89-D(4):743–754, April 2006. (In Japanese).

[13] Y. Hasegawa, S. Abe, H. Matsutani, K. Anjo, T. Awashima, and H. Amano. An Adaptive Cryptographic Accelerator for IPsec on Dynamically Reconfigurable Processor. In *Proc. of the 4th IEEE Int'l Conf. on Field Programmable Technology (ICFPT2005)*, pages 163–170, December 2005.

[14] S. Abe, Y. Hasegawa, T. Toi, T. Inuo, and H. Amano. Adaptive Computing on the Dynamically Reconfigurable Processor. In *Proc. of the 9th IEEE Symp. on Low-Power and High-Speed Chips (COOL Chips IX)*, pages 409–421, April 2006.

[15] S. Abe, Y. Hasegawa, T. Toi, T. Inuo, and H. Amano. An Adaptive Viterbi Decoder on the Dynamically Reconfigurable Processor. In *Proc. of the 5th IEEE Int'l Conf. on Field Programmable Technology (ICFPT2006)*, pages 285–288, December 2006.

[16] Y. Hasegawa, H. Matsutani, M. Koibuchi, , and H. Amano. Reconfigurable Architectures with On-Chip Networks for Multitask Designs. In *IEICE Technical Reports, RECONF2006-5*, pages 25–31, May 2006. (In Japanese).

[17] Y. Hasegawa, S. Tsutsumi, T. Nakamura, T. Nishimura, T. Sano, M. Kato, S. Saito, , and H. Amano. Design and Implementation of the Dynamically Reconfigurable Processor MuCCRA-1. In *Proc. of the 5th Symp. on Advanced Computing Systems and Infrastructures (SACSIS2006)*, pages 95–102, May 2007. (In Japanese).

[18] Y. Hasegawa, S. Tsutsumi, V. Tunbunheng, A. Parimala, T. Nakamura, T. Nishimura, and H. Amano. Design Methodology and Trade-offs Analysis for Parameterized Dynamically Reconfigurable Processor Arrays. In *Proc. of the 17th IEEE Int'l Conf. on Field-Programmable Logic and Application (FPL2007)*, August 2007. (to appear).

[19] Xilinx, Inc. `http://www.xilinx.com/`.

[20] Altera Corporation. `http://www.altera.com/`.

[21] E. Ahmed and J. Rose. The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and Density. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 12, 2004.

[22] G. Lemieux and D. Lewis. *Design of Interconnection Networks for Programmable Logic*. Kluwer Academic Publishers, 2004.

[23] Xilinx, Inc. Spartan-3 Generation FPGA User Guide (v1.2).

[24] S. Goldstein, H. Schmit, M. Moe, M. Budiuy, S. Cadambi, R. Taylor, and R. Laufer. PipeRench: A Coprocessor for Streaming Multimedia Acceleration. In *Proc. of the 26th Annual Int'l Symp. on Computer Architecture (ISCA1999)*, pages 28–39, May 1999.

[25] J. Arnold, D. Buell, and E. Davis. Splash 2. In *Proc. of the 4th Annual ACM Symp. on Parallel Algorithms and Architectures*, pages 316–322, June 1992.

[26] J. Arnold and W. Kleinfelder. *Splash 2: FPGAs in a Custom Computing Machine*. IEEE Computer Society, 1996.

[27] M. Numa. Reconfigurable Machines: RM-I to RM-IV and Their Applications. In *IEICE Technical Reports, VLD*, pages 119–126, December 2006. (In Japanese).

[28] T. Yamaguchi, S. Annzai, A. Mizutani, N. Kuroki, and M. Numa. Reconfigurable Machine: RM-V and Its Applications. In *IPSJ Technical Reports, 2000-SLDM-96-6*, pages 33–39, November 2000. (In Japanese).

[29] K. Nakajima, H. Sato, H. Asami, M. Iida, K. Shindome, H. Mori, K. Takahashi, and Y. Mizukami. FPGA-based Parallel Machine : RASH. In *Proc. of the 20th Int'l Conf. on Applied Informatics (AI2002)*, pages 269–273, February 2002.

[30] R. Amerson, R. J. Carter, W. B. Culbertson, P. Kuekes, and G. Snider. Teramac – Configurable Custom Computing. In *Proc. of the 3rd IEEE Workshop on FPGAs for Custom Computing Machines (FCCM1995)*, pages 32–38, April 1995.

[31] J. R. Hauser and J. Wawrzynek. Garp: A MIPS Processor with a Reconfigurable Coprocessor. In *Proc. of the 5th IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM1997)*, pages 12–21, April 1997.

[32] T. J. Callahan, J. R. Hauser, and J. Wawrzynek. The Garp Architecture and C Compiler. *IEEE Computer*, 33(4):62–69, April 2000.

[33] C. R. Rupp, M. Landguth, T. Garverick, E. Gomersall, H. Holt, J. M. Arnold, and M. Gokhale. The NAPA Adaptive Processing Architecture. In *Proc. 6th IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM1998)*, pages 28–37, April 1998.

[34] P. M. Athanas and H. F. Silverman. Processor Reconfiguration Through Instruction-Set Metamorphosis. *IEEE Computer*, 26(3):11–18, March 1993.

[35] M. Wazlowski, L. Agarwal, T. Lee, A. Smith, E. Lam, P. Athanas, H. Silverman, , and S. Ghosh. PRISM-II Compiler and Architecture. In *Proc. of the 1st IEEE Workshop on FPGAs for Custom Computing Machines (FCCM1993)*, pages 9–16, April 1993.

[36] F. Raimbault, D. Lavenier, S. Rubini, and B. Pottier. Fine Grain Parallelism on a MIMD Machine Using FPGAs. In *Proc. of the 1st IEEE Workshop on FPGAs for Custom Computing Machines (FCCM1993)*, pages 2–8, April 1993.

[37] R. Razdan and M. Smith. A High Performance Microarchitecture with Hardware Programmable Functional Units. In *Proc. of the 27th Annual IEEE/ACM Int'l Symp. on Microarchitecture (MICRO-27)*, pages 172–180, 1994.

[38] R. Witting and P. Chow. OneChip: An FPGA Processor with Reconfigurable Logic. In *Proc. of the 4th IEEE Symp. on FPGAs for Custom Computing Machines (FCCM1996)*, pages 126–135, 1996.

[39] S. Hauck, T. Fry, M. Hosler, and J. Kao. The Chimaera Reconfigurable Functional Unit. In *Proc. of the 5th IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM1997)*, pages 87–96, April 1997.

[40] Z. Ye, A. Moshovos, S. Hauck, and P. Banerjee. CHIMAERA: A High-Performance Architecture with a Tightly-Coupled Reconfigurable Functional Unit. In *Proc. of the 27th Annual Int'l Symp. on Computer Architecture (ISCA2000)*, pages 225–235, 2000.

[41] R. E. Gonzalez. Xtensa: A Configurable and Extensible Processor. *IEEE Micro*, 20(2):60–70, March 2000.

[42] Tensilica, Inc. `http://www.tensilica.com/`.

[43] A. Mizuno, L. Kohno, R. Ohyama, T. Tokuyoshi, H. Uetani, H. Eichel, T. Miyamori, N. Matsumoto, and M. Matsui. Design Methodology and System for a Configurable Media Embedded Processor Extensible to VLIW Architecture. In *Proc of the 20th IEEE Int'l Conf. on Computer Design: VLSI in Computers and Processors (ICCD2002)*, pages 2–7, October 2002.

[44] S. Ishiwata, T. Yamakage, Y. Tsuboi, T. Shimazawa, T. Kitazawa, S. Michinaka, K. Yahagi, H. Takeda, A. Oue, T. Kodama, N. Matsumoto, T. Kamei, M. Saito, T. Miyamori, G. Ootomo, and M. Matsui. A Single-chip MPEG-2 Codec Based on Customizable Media Embedded Processor. *IEEE Journal of Solid-State Circuits*, 38(3):530–540, March 2003.

[45] T. Kiyohara. Multimedia Processor-based Platform for a Wide Range of Digital Consumer Electronics. In *Proc. of the 8th IEEE Symp. on Low-Power and High Speed Chips (COOL Chips XIII)*, pages 133–141, April 2005.

[46] M. Nakajima, T. Yamamoto, M. Yamasaki, T. Hosoki, and M. Sumital. Low Power Techniques for Mobile Application SoCs Based on Integrated Platform UniPhier. In *Proc. of the 12th Asia and South Pacific Design Automation Conf. (ASPDAC2007)*, pages 649–653, January 2007.

[47] I. Kuon and J. Rose. Measuring the Gap Between FPGAs and ASICs. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 26(2):203–215, February 2007.

[48] K. Compton, J. Cooley, S. Knol, and S. Hauck. Configuration Relocation and Defragmentation for Run-Time Reconfigurable Computing. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 10(3):209–220, June 2002.

[49] S. Trimberger, D. Carberry, A. Johnson, and J. Wong. A Time-Multiplexed FPGA. In *Proc. of the 5th IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM1997)*, pages 22–28, April 1997.

[50] E. Tau, I. Eslick, D. Chen, J. Brown, and A. DeHon. A First Generation DPGA Implementation. In *Proc. of the 3rd Canadian Workshop on Field-Programmable Devices*, May 1995.

[51] X. P. Ling and H. Amano. WASMII: A Data Driven Computer on a Virtual Hardware. In *Proc. of the 1st IEEE Symp. on FPGAs for Custom Computing Machines (FCCM1993)*, pages 33–42, April 1993.

[52] Y. Shibata, M. Uno, H. Amano, K. Furuta, T. Fujii, and M. Motomura. A Virtual Hardware System on a Dynamically Reconfigurable Logic Device. In *Proc. of the 8th IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM2000)*, pages 295–296, April 2000.

[53] T. Toi, N. Nakamura, Y. Kato, T. Awashima, K. Wakabayashi, and L. Jing. High-level Synthesis Challenges and Solutions for a Dynamically Reconfigurable Processor. In *Proc. of the 2006 IEEE/ACM Int'l Conf. on Computer-Aided Design (ICCAD2006)*, pages 702–708, November 2006.

[54] K. Wakabayashi and T. Okamoto. C-Based SoC Design Flow and EDA Tools: An ASIC and System Vendor Perspective. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 19(12):1507–1522, December 2000.

[55] K. Tanigawa, T. Hironaka, A. Kojima, and N. Yoshida. PARS Architecture: A Reconfigurable Architecture with Generalized Execution Model–Design and Implementation of Its Prototype Processor. *IEICE Trans. on Information & Systems*, E86-D(5):830–840, May 2003.

[56] H. Singh, M.-H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. Chaves Filho. MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Computation-Intensive Applications. *IEEE Trans. on Computers*, 49(5):465–481, May 2000.

[57] D. Kissler, F. Hannig, A. Kupriyanov, and J. Teich. A Highly Parameterizable Parallel Processor Array Architecture. In *Proc. of the 5th IEEE Int'l Conf. on Field Programmable Technology (ICFPT2006)*, pages 105–112, December 2006.

[58] Y. Kurose, I. Kumata, M. Okabe, H. Hanaki, K. Seno, K. Hasegawa, H. Ozawa, S. Horiike, T. Wada, S. Arima, K. Taniguchi, K. Ono, H. Hokazono, T. Hiroi, T. Hirano, and S. Takashima. A 90nm Embedded DRAM Single Chip LSI with a 3D Graphics, H.264 Codec Engine, and a Reconfigurable Processor. In *Proc. of the 16th Symp. on High Performance Chips (Hot Chips 16)*, August 2004.

[59] Sony Corporation. `http://www.sony.net/Products/SC-HP/cx_news/vol42/pdf/sideview42.pdf`.

[60] H. Ito, R. Konishi, H. Nakada, K. Oguri, A. Nagoya, N. Imlig, K. Nagami, T. Shiozawa, and M. Inamori. Dynamically Reconfigurable Logic LSI: PCA-1. In *Proc. of the 14th Symp. on VLSI Circuits*, pages 103–106, June 2001.

[61] H. Ito, R. Konishi, H. Nakada, H. Tsuboi, Y. Okuyama, and A. Nagoya. Dynamically Reconfigurable Logic LSI: PCA-2. *IEICE Trans. on Information & System*, E87-D(8):2011–2020, August 2004.

[62] B. Salefski and L. Caglar. Re-Configurable Computing in Wireless. In *Proc. of the 38th Design Automation Conference (DAC2001)*, pages 178–183, June 2001.

[63] M. Motomura. A Dynamically Reconfigurable Processor Architecture. In *Proc. of Microprocessor Forum*, October 2002.

[64] T. Sugawara, K. Ide, and T. Sato. Dynamically Reconfigurable Processor Implemented with IPFlex's DAPDNA Technology. *IEICE Trans. on Information & System*, E87-D(8):1997–2003, August 2004.

[65] T. Sato. Dynamic Reconfiguration and Its Granularity inside Future DAPDNA Architecture. In *Proc. of Int'l Symp. on Advanced Reconfigurable Systems*, pages 114–127, December 2005.

[66] T. Sato. A Dual-Core Dynamically Reconfigurable Engine Employs 955 Parallel Processing Elements. In *Proc. of Microprocessor Forum*, May 2007.

[67] T. Komada, T. Tsunoda, M. Takeda, H. Tanaka, Y. Akita, M. Sato, and M. Ito. Flexibile Engine: A Dynamic Reconfigurable Accelerator with High Performance and Low Power Consumption. In *Proc. of the 9th IEEE Symp. on Low-Power and High Speed Chips (COOL Chips IX)*, pages 393–408, April 2006.

[68] F. Veredas, M. Scheppler, W. Moffat, and B. Mei. Custom Implementation of the Coarse-Grained Reconfigurable ADRES Architecture for Multimedia Purposes. In *Proc. of 15th IEEE Int'l Conf. on Field Programmable Logic and Application (FPL)*, pages 106–111, August 2005.

[69] M. Saito, H. Fujisawa, N. Ujiie, and H. Yoshizawa. Cluster Architecture for Reconfigurable Signal Processing Engine for Wireless Communication. In *Proc. of the 15th IEEE Int'l Conf. on Field Programmable Logic and Applications (FPL2005)*, pages 353–359, August 2005.

[70] T. Stansfield. Using Multiplexers for Control and Data in D-Fabrix. In *Proc. of the 13th Int'l Conf. on Field Programmable Logic and Applications (FPL2003)*, pages 416–425, August 2003.

[71] A. Marshall, T. Stansfield, I. Kostarnov, J. Vuillemin, and B. Hutchings. A Reconfigurable Arithmeric Array for Multimedia Applications. In *Proc. of the ACM/SIGDA 7th Int'l Symp. on Field-Programmable Gate Arrays (FPGA1999)*, pages 135–143, February 1999.

[72] T. Matsumoto, K. Kimura, H. Takano, T. Amatsubo, K. Mori, K. Senda, S. Inoue, and M. Matsui. Performance Evaluation of Reconfigurable Processing Array in Area Efficiency and Operating Frequency. In *Proc. of the 9th IEEE Symp. on Low-Power and High Speed Chips (COOL Chips IX)*, pages 423–434, April 2006.

[73] B. Levine. Kilocore: Scalable, High-Performance, and Power Efficient Coarse-grained Reconfigurable Fabrics. In *Proc. of Int'l Symp. on Advanced Reconfigurable Systems*, pages 129–158, December 2005.

[74] H. Schmit, D. Whelihan, A. Tsai, M. Moe, B. Levine, and R. R. Taylor. PipeRench: A Virtualized Programmable Datapath in 0.18 Micron Technology. In *Proc. of the 24th IEEE Custom Integrated Circuits Conf. (CICC2002)*, pages 63–66, May 2002.

[75] PACT Inc. *XPP-III Processor Overview*, version 2.0 edition, July 2006.

[76] PACT Inc. *Video Decoding on XPP-III*, revision 1.1 edition, July 2006.

[77] Stretch Inc. `http://www.stretchinc.com/_files/s6ArchitectureOverview.pdf`.

[78] J. Arnold. S5: The Architecture and Development Flow of a Software Configurable Processor. In *Proc. of the 4th IEEE Int'l Conf. on Field Programmable Technology (ICFPT2005)*, pages 120–128, December 2005.

[79] A. DeHon. Reconfigurable Architectures for General-Purpose Computing. *AI Technical Report 1586*, September 1996.

[80] M. Suzuki, Y. Hasegawa, Y. Yamada, N. Kaneko, K. Deguchi, H. Amano, K. Anjo, M. Motomura, K. Wakabayashi, T. Toi, and T. Awashima. Stream Applications on the Dynamically Reconfigurable Processor. In *Proc. of the 3rd IEEE Int'l Conf. on Field Programmable Technology (ICFPT2004)*, pages 137–144, December 2004.

[81] S. Kurotaki, N. Suzuki, K. Nakadai, H. G. Okuno, and H. Amano. Implementation of Active Direction-Pass Filter on Dynamically Reconfigurable Processor. In *Proc. of the IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS2005)*, pages 215–220, August 2005.

[82] S. Kurotaki, N. Suzuki, K. Nakadai, H. Okuno, and H. Amano. Implementation and Evaluation of Sound Source Separation Filter on Dynamically Reconfigurable Processor. *IEICE Trans. on Information & Systems*, J90-D(3):897–907, March 2007.

[83] U. J. Kapasi, S. Rixner, W. J. Dally, B. Khailany, J. H. Ahn, P. Mattson, and J. D. Owens. Programmable Stream Processors. *IEEE Computer*, 36(8):54–62, August 2003.

[84] V. M. Tuan, Y. Hasegawa, and H. Amano. Performance Analysis of Multi-process Execution Model on Dynamically Reconfigurable Processor. In *Proc. of the 7th Int'l Conf. on Engineering of Reconfigurable Systems & Algorithms (ERSA2007)*, pages 203–206, June 2007.

[85] S.Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401, 1998.

[86] K. Anjo, A. Okamura, and M. Motomura. Wrapper-based Bus Implementation Techniques for Performance Improvement and Cost Reduction. *IEEE Journal of Solid State Circuits*, 36(5):804–817, May 2004.

[87] The Keyed-Hash Message Authentication Code (HMAC). Federal Information Processing Standard (FIPS) Publication 198, 2002.

[88] Advanced Encryption Standard (AES). Federal Information Processing Standard (FIPS) Publication 197, 2001.

[89] P. Chodowiec, K. Gaj, P. Bellows, and B. Schott. Experimental Testing of the Gigabit IPSec-Compliant Implementations of Rijndael and Triple DES Using SLAAC-1V FPGA Accelerator Board. In *Proc. of the 4th Int'l Conf. on Information Security (ISC2001)*, pages 220–234, October 2001.

[90] A. Dandails, V. K. Prasanna, and J. D. P. Rolim. An Adaptive Cryptographic Engine for IPsec Architectures. In *Proc. of the 8th IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM2000)*, pages 132–141, April 2000.

[91] R. Tessier, S. Swaminathan, R. Ramaswamy, D. Goeckel, and W. Burleson. A Reconfigurable, Power-Efficient Adaptive Viterbi Decoder. *IEEE Trans. on Very Large Scale Integrated Systems*, 13(4):484–488, April 2005.

[92] H. Amano, A. Jouraku, and K. Anjo. A Dynamically Adaptive Hardware on Dynamically Reconfigurable Processor. *IEICE Trans. on Communications*, E86-B(12):3385–3391, December 2003.

[93] T. K. Truong, M. T. Shih, I. S. Reed, and E. H. Satorius. A VLSI Design for A Trace-Back Viterbi Decoder. *IEEE Trans. on Communications*, 40:616–624, March 1992.

[94] E. Yeo, S. Augsburger, W. R. Davis, and B. Nikolic. Implementation of High Throughput Soft Output Viterbi Decoders. In *Proc of the IEEE Workshop on Signal Processing Systems (SIPS2002)*, pages 146–151, October 2002.

[95] T. S. Rappaport. *Wireless Communications: Principles and Practice*. Prentice Hall, 1996.

[96] T. Marescaux, A. Bartic, D. Verkest, S. Vernalde, and R. Lauwereins. Interconnection Networks Enable Fine-Grain Dynamic Multitasking on FPGAs. In *Proc. of the 12th Int'l Conf. on Field Programmable Logic and Applications (FPL2002)*, pages 795–805, September 2002.

[97] W. J. Dally and B. Towles. Route Packets, Not Wires: On-Chip Interconnection Networks. In *Proc. of the 38th Design Automation Conf. (DAC2001)*, pages 684–689, June 2001.

[98] L. Benini and G. D Micheli. Networks on Chips: A New SoC Paradigm. *IEEE Computer*, 35(1):70–78, January 2002.

[99] Y. Yamada, H. Amano, M. Koibuchi, A. Jouraku, K. Anjo, and K. Nishimura. Folded Fat H-Tree: An Interconnection Topology for Dynamically Reconfigurable Processor Array. In *Proc. of the IFIP Int'l Conf. on Embedded and Ubiquitous Computing (EUC2004)*, pages 301–311, August 2004.

[100] C. J. Glass and L. M. Ni. The Turn Model for Adaptive Routing. In *Proc. of the 19th Annual Int'l Symp. on Computer Architecture (ISCA1992)*, pages 278–287, May 1992.

[101] H. Amano, T. Inuo, H. Kami, T. Fujii, and M. Suzuki. Techniques for Virtual Hardware on a Dynamic Reconfigurable Processor -An approach to tough cases-. In *Proc. of the 14th Int'l Conf. on Field Programmable Logic and Applications (FPL2004)*, pages 464–473, September 2004.

[102] V. Tunbunheng, M. Suzuki, and H. Amano. RoMultiC: Fast and Simple Configuration Data Multicasting Scheme for Coarse Grain Reconfigurable Devices. In *Proc. of the 4th IEEE Int'l Conf. on Field Programmable Technology (ICFPT2005)*, pages 129–136, Dec 2005.

[103] H. Amano, S. Abe, K. Deguchi, and Y. Hasegawa. An I/O Mechanism on a Dynamically Reconfigurable Processor - Which should be moved: Data or Configuration. In *Proc. of the 15th IEEE Int'l Conf. on Field Programmable Logic and Applications (FPL2005)*, pages 347–352, September 2005.

[104] S. Tsutsumi, V. Tunbungheng, Y. Hasegawa, H. Matsutani, A. Parimala, T. Nakamura, T. Nishimura, M. Kato, S. Saito, T. Sano, N. Seki, K. Hirai, M. K. Yi, and H. Amano. Scheduling Algorithms for

Multicast Configuration. In *IEICE Technical Reports, RECONF2006-73*, pages 49–54, January 2007. (In Japanese).

[105] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal. The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs. *IEEE Micro*, 22(2):25–35, 2002.

[106] W. R. Davis, J. Wilson, S. Mick, J. Xu, H. Hua, C. Mineo, A. M. Sule, M. Steer, and P. D. Franzon. Demystifying 3D ICs: The Pros and Cons of Going Vertical. *IEEE Design and Test of Computers*, 22(6):498–510, November 2005.

[107] S. Saito, Y. Hasegawa, Y. Kohama, Y. Sugimori, and H. Amano. 3-D Dynamically Reconfigurable Device using Inter-Chip Wireless Communication: MuCCRA-Cube. In *IEICE Technical Reports, RECONF2007-5*, pages 25–30, May 2007. (In Japanese).

[108] C. T. Chow, L. S. M. Tsui, P. H. W. Leong, W. Luk, and S. J. E. Wilton. Dynamic Voltage Scaling for Commercial FPGAs. In *Proc. of the 4th IEEE Int'l Conf. on Field Programmable Technology (ICFPT2005)*, pages 173–180, December 2005.

[109] Y. Lin, F. Li, and L. He. Circuits and Architecture Evaluation for Field Programmable Gate Array with Configurable Supply Voltage. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 13(9):1035–1046, September 2005.

[110] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose. Microarchitectural Techniques for Power Gating of Execution Units. In *Proc. of the 9th Int'l Symp. on Low Power Electronics and Design (ISLPED2004)*, pages 32–37, August 2004.

Publications

Journal Papers

[1] Satoshi Tsutsumi, Hideharu Amano, Yohei Hasegawa, Kenichiro Ishikawa, Shohei Abe, Syunsuke Kurotaki, Takuro Nakamura, and Takashi Nishimura, A Clock Control Mechanism Using Context Dependent Delay for Dynamically Reconfigurable Processors, *IEICE Trans. on Information & Systems*, 2007. (to appear) (In Japanese).

[2] Yohei Hasegawa, Shohei Abe, Syunsuke Kurotaki, Vu Manh Tuan, and Hideharu Amano, Evaluation of Time-multiplexed Execution on the Dynamically Reconfigurable Processor,*IPSJ Trans. on Advanced Computing Systems (ACS)*, Vol. 47, No. SIG12 (ACS15), pp. 171-181, September 2006. (In Japanese)

[3] Masayasu Suzuki, Yohei Hasegawa, Shohei Abe, Vu Manh Tuan, and Hideharu Amano, A Novel Cost-Effective Context Memory Structure for Dynamically Reconfigurable Processors, *IEICE Trans. on Information & Systems*, Vol. J89-D, No. 6, pp. 1101-1109, June 2006. (In Japanese)

[4] Yohei Hasegawa, Shohei Abe, Hiroki Matutani, Kenichiro Anjo, Toru Awashima, and Hideharu Amano, An Adaptive Cryptographic Accelerator for IPsec on the Dynamically Reconfigurable Processor, *IEICE Trans. on Information & Systems*, Vol. J89-D, No. 4, pp. 743-754, April 2006. (In Japanese)

International Conference Papers

[5] Yohei Hasegawa, Tsutsumi Satoshi, Vasutan Tunbunheng, Takuro Nakamura, Takashi Nishimura and Hedeharu Amano. Design Methodology and Trade-offs Analysis for Parameterized Dynamically Reconfigurable Processor Arrays, *In Proc. of the 17th IEEE Int'l Conf. on Field Programmable Logic and Applications (FPL2007)*, Amsterdam, Netherlands, August 2007.

[6] Vu Manh Tuan, Yohei Hasegawa, and Hideharu Amano. Performance Analysis of Multi-Process Execution Model on Dynamically Reconfigurable Processor, *In Proc. of the 7th Int'l Conf. on Engineering of Reconfigurable Systems and Algorithms (ERSA2007)*, pp. 203-206, Las Vegas, NV, June 2007

[7] Takashi Nishimura, Keiichiro Hirai, Seidai Takeda, Yohei Hasegawa, Satoshi Tsutsumi, Hedeharu Amano, Kimiyoshi Usami. Power Reduction Technique for Dynamically Reconfigurable Processor, *In Proc. of the 10th IEEE Symp. on Low-Power and High-Speed Chips (COOL Chips X)*, Yokohama, Japan, April 2007.

[8] Shohei Abe, Yohei Hasegawa, Takao Toi, Takeshi Inuo, and Hideharu Amano. An Adaptive Viterbi Decoder on the Dynamically Reconfigurable Processor, *In Proc. of the 5th IEEE Int'l Conf. on Field Programmable Technology (ICFPT2006)*, pp. 285-288, Bangkok, Thailand, December 2006.

[9] Hideharu Amano, Yohei Hasegawa, Shohei Abe, Kenichiro Ishikawa, Syunsuke Kurotaki, Takuro Nakamura, and Takashi Nishimura. A Context Dependent Clock Control Mechanism for Dynamically Reconfigurable Processors, *In Proc. of the 16th Int'l Conf. on Field Programmable Logic and Applications (FPL2006)*, pp. 575-580, Madrid, Spain, August 2006.

[10] Yohei Hasegawa, Shohei Abe, Shunsuke Kurotaki, Vu Manh Tuan, Naohiro Katsura, Takuro Nakamura, Takashi Nishimura, and Hideharu Amano. Performance and Power Analysis of Time-multiplexed Execution on Dynamically Reconfigurable Processor, *In Proc. of the 20th Int'l Parallel and Distributed Processing Symp. (IPDPS 2006) / Reconfigurable Architectures Workshop (RAW2006)*, Rhodes Island, Greece, April 2006.

[11] Masayasu Suzuki, Yohei Hasegawa, Vu Manh Tuan, Shohei Abe, and Hideharu Amano. A Cost Effective Context Memory Structure for Dynamically Reconfigurable Processors, *In Proc. of the 20th Int'l Parallel and Distributed Processing Symp. (IPDPS 2006) / Reconfigurable Architectures Workshop (RAW2006)*, Rhodes Island, Greece, April 2006.

[12] Shohei Abe, Yohei Hasegawa, Takao Toi, Takeshi Inuo, and Hideharu Amano. Adaptive Computing on the Dynamically Reconfigurable Processor, *In Proc. of Int'l Conf. on Cool Chips IX*, pp. 409-421, Yokohama, Japan, April 2006.

[13] Vu Manh Tuan, Yohei Hasegawa, Naohiro Katsura, and Hideharu Amano. Performance-Cost Trade-off Evaluation for the DCT Implementation on the Dynamically Reconfigurable Processor, *In Proc. of Int'l Workshop on Applied Reconfigurable Computing (ARC2006)*, Delft, Netherlands, March 2006.

[14] Yohei Hasegawa, Shohei Abe, Shunsuke Kurotaki, Vu Manh Tuan, Naohiro Katsura, Takuro Nakamura, Takashi Nishimura, and Hideharu Amano. Application-Based Performance and Power Analysis of Dynamically Reconfigurable Processor. *Int'l Symp. on Advanced Reconfigurable Systems*, Kyoto, Japan, December 2005.

[15] Yohei Hasegawa, Shohei Abe, Hiroki Matsutani, Kenichiro Anjo, Toru Awashima, and Hideharu Amano. An Adaptive Cryptographic Accelerator for IPsec on Dynamically Reconfigurable Processor, *In Proc. of the 4th IEEE Int'l Conf. on Field Programmable Technology (ICFPT2005)*, pp. 163-170, Singapore, December 2005.

[16] Hideharu Amano, Shohei Abe, Katsuaki Deguchi, and Yohei Hasegawa. An I/O Mechanism on a Dynamically Reconfigurable Processor -Which should be moved: Data or Configuration? -, *In Proc. of the 15th Int'l Conf. on Field Programmable Logic and Applications (FPL2005)*, pp. 347-352, Tampere, Finland, August 2005.

[17] Hideharu Amano, Shohei Abe, Yohei Hasegawa, Katsuaki Deguchi, and Masayasu Suzuki. Performance and Cost Analysis of Time-multiplexed Execution on the Dynamically Reconfigurable Processor, *In Proc. of the 13th IEEE Symp. on Field-programmable Custom Computing Machines (FCCM2005)*, Napa, CA, April 2005.

[18] Shohei Abe, Yohei Hasegawa, and Hideharu Amano. Implementation of AES on the Dynamic Reconfigurable Processor, *In Proc. of the 8th IEEE Symp. on Low-Power and High-Speed Chips (COOL Chips VIII)*, Yokohama, Japan, April 2005.

[19] Yohei Hasegawa, Shohei Abe, Katsuaki Deguchi, Masayasu Suzuki, and Hideharu Amano. Time-Multiplexed Execution on the Dynamically Reconfigurable Processor: A Performance-Cost Evaluation, *In Proc. of the 13th ACM/SIGDA Int'l Symp. on Field Programmable Gate Arrays (FPGA2005)*, p. 265, Monterey, CA, February 2005.

[20] Masayasu Suzuki, Yohei Hasegawa, Yutaka Yamada, Naoto Kaneko, Katsuaki Deguchi, Hideharu Amano, Kenichiro Anjo, Masato Motomura, Kazutoshi Wakabayashi, Takeo Toi, and Toru Awashima. Stream Applications on the Dynamically Reconfigurable Processor, *In Proc. of the 3rd IEEE Int'l Conf. on Field Programmable Technology (ICFPT2004)*, pp. 137-144, Brisbane, Australia, December 2004.

[21] Masayasu Suzuki, Yohei Hasegawa, Yutaka Yamada, Katsuaki Deguchi, Kenichiro Anjo, Toru Awashima, and Hideharu Amano. Stream Application Evaluation on the DRP-1, *In Proc. of the 7th IEEE Symp. on Low-Power and High-Speed Chips (COOL Chips VII)*, pp. 33-47, Yokohama, Japan, April 2004.

[22] Noriaki Suzuki, Syunsuke Kurotaki, Masayasu Suzuki, Naoto Kaneko, Yutaka Yamada, Katsuaki Deguchi, Yohei Hasegawa, Hideharu Amano, Kenichiro Anjo, Masato Motomura, Kazutoshi Wakabayashi, Takeo Toi, and Toru Awashima. Implementing and Evaluating Stream Applications on the Dynamically Reconfigurable Processor, *In Proc. of the 12th IEEE Int'l Conf. on Field Programmable Custom Computing Machines (FCCM2004)*, pp. 328-329, Napa, CA, April 2004.

Domestic Conference Papers and Technical Reports

Original Papers

[23] Yohei Hasegawa, Satoshi Tsutsumi, Takuro Nakamura, Takashi Nishimura, Toru Sano, Masaru Kato, Shotaro Saito, and Hideharu Amano. Design and Implementation of the Dynamically

Reconfigurable Processor MuCCRA-1, *The 5th Symp. on Advanced Computing Systems and Infrastructures (SACSIS2007)*, pp. 95-102, May 2007, ***Best Young Researcher Award, IEEE Computer Society Japan Chapter Award***. (In Japanese)

[24] Yohei Hasegawa, Shohei Abe, Syunsuke Kurotaki, Vu Manh Tuan, and Hideharu Amano. Evaluation of Time-multiplexed Execution on the Dynamically Reconfigurable Processor, *The 4th Symp. on Advanced Computing Systems and Infrastructures (SACSIS2006)*, pp. 135-142, May 2006, ***Best Young Researcher Award, IEEE Computer Society Japan Chapter Award***. (In Japanese)

[25] Yohei Hasegawa, Hiroki Matutani, Michihiro Koibuchi, and Hideharu Amano. Reconfigurable Architectures with On-Chip Networks for Multitask Designs, *IEICE Technical Reports, RECONF2006-5*, pp. 25-31, May 2006. (In Japanese)

[26] Yohei Hasegawa, Takashi Nishimura, Shohei Abe, Syunsuke Kurotaki, Vu Manh Tuan, and Hideharu Amano. Application-Based Performance and Power Analysis on Dynamically Reconfigurable Processor, *The 27th Parthenon Symp.*, pp. 3-10, December 2005. (In Japanese)

[27] Yohei Hasegawa, Hideharu Amano, Shohei Abe, Syunsuke Kurotaki, and Vu Manh Tuan, Performance and Power Analysis of Time-multiplexed Execution on Dynamically Reconfigurable Processor, *IEICE Technical Reports, RECONF2005-35*, pp. 31-36, September 2005. (In Japanese)

[28] Yohei Hasegawa, Shohei Abe, Hiroki Matutani, Kenichiro Anjo, Toru Awashima, and Hideharu Amano. Switchable Multi-Cryptographic Engines for Embedded Network Security on a Dynamically Reconfigurable Processor, *IEICE Technical Reports, CPSY2004-92*, pp. 13-18, March 2005. (In Japanese)

[29] Yohei Hasegawa, Shohei Abe, Kenichiro Anjo, Toru Awashima, and Hideharu Amano. The Design and Implementation of the IPsec Accelerator with the Dynamically Reconfigurable Processor, *IEICE Technical Reports, CPSY2004-37*, pp. 29-34, December 2004. (In Japanese)

[30] Yohei Hasegawa, Yutaka Yamada, Katsuaki Deguchi, Kenichiro Anjo, Toru Awashima, and Hideharu Amano. The Implementation of the Block Cipher RC6 on the Reconfigurable Processor, *IEICE Technical Reports, CPSY2003-119*, pp. 29-34, January 2004. (In Japanese)

Joint Papers

[31] Masaru Kato, Yohei Hasegawa, and Hideharu Amano. Evaluation of MuCCRA-D: A Dynamically Reconfigurable Processor with Directly Interconnected PEs, In *Proc. of the 30th Parthenon Symp.*, pp. 1-7, June 2007. (In Japanese)

[32] Naomi Seki, Yohei Hasegawa, Hideharu Amano, Naoaki Ohkubo, Seidai Takeda, Toshihiro Kashima, Toshiaki Shirai, Kimiyoshi Usami, Masaaki Kondo, Hiroshi Nakamura. A Fine

Grain Dynamic Sleep Control Scheme in MIPS R3000, *IPSJ Technical Reports, 2007-ARC-173(9)*, pp. 49-54, May 2007. (In Japanese)

[33] Shotato Saito, Yohei Hasegawa, Yoshinori Kohama, Yasufumi Sugimori, and Hideharu Amano. 3-D Dynamically Reconfigurable Device using Inter-Chip Wireless Communication MuCCRA -Cube, *IEICE Technical Reports, RECONF2006-5*, pp. 25-30, May 2007. (In Japanese)

[34] Toru Sano, Takuro Nakamura, Satoshi Tsutsumi, Yohei Hasegawa, and Hideharu Amano. Techniques to Decrease the Configuration Data Transfer Time in Dynamically Reconfigurable Processor MuCCRA, *IEICE Technical Reports, RECONF2006-10*, pp. 55-60, May 2007. (In Japanese)

[35] Masaru Kato, Yohei Hasegawa, and Hideharu Amano. MuCCRA-D: A Dynamically Reconfigurable Processor with Directly Interconnected PEs, *IEICE Tech. Reports, RECONF2006-12*, pp. 67-72, May 2007. (In Japanese)

[36] Keiichiro Hirai, Seidai Takeda, Takashi Nishimura, Yohei Hasegawa, Satoshi Tsutsumi, Kimiyoshi Usami, and Hideharu Amano. Power Reduction of Dynamically Reconfigurable Processor MuCCRA, *IEICE Tech. Reports, RECONF2007-11*, pp. 61-66, May 2007. (In Japanese)

[37] Satoshi Tsutsumi, Vasutan Tunbungheng, Yohei Hasegawa, Hiroki Matsutani, Adepu Parimala, Takuro Nakamura, Takashi Nishimura, Masaru Kato, Shotaro Saito, Toru Sano, Naomi Seki, Keiichiro Hirai, Mao Kai Yi, and Hideharu Amano. Scheduling Algorithms for Multicast Configuration, *IEICE Technical Reports, RECONF2006-73*, pp. 49-54, January 2007. (In Japanese)

[38] Takuro Nakamura, Yohei Hasegawa, Satoshi Tsutsumi, Hiroki Matsutani, Vasutan Tunbungheng, Adepu Parimala, Takashi Nishimura, Masaru Kato, Shotaro Saito, Toru Sano, Naomi Seki, Keiichiro Hirai, Mao Kai Yi, and Hideharu Amano. Implementation of Dynamically Reconfigurable Processor MuCCRA, *IEICE Technical Reports, RECONF2006-72*, pp. 43-48, January 2007. (In Japanese)

[39] Hideharu Amano, Yohei Hasegawa, Satoshi Tsutsumi, Takuro Nakamura, Takashi Nishimura, Vasutan Tunbungheng, Adepu Parimala, Masaru Kato, Toru Sano, Shotaro Saito, and Hiroki Matsutani. MuCCRA-1: A Dynamically Reconfigurable Processor Prototype Chip, In *Proc of the 29th Parthenon Symp.*, pp. XX-XX, December 2006. (In Japanese)

[40] Naohiro Katsura, Yohei Hasegawa, Vu Manh Tuan, Hiroki Matsutani, and Hideharu Amano. Performance Evaluation of Multi-core DRP for Stream Application, *IEICE Technical Reports, RECONF2006-52*, pp. 49-54, November 2006. (In Japanese)

[41] Hideharu Amano, Yohei Hasegawa, Takuro Nakamura, Takashi Nishimura, and Vasutan Tunbungheng. A Configuration Control Mechanism for a Dynamically Reconfigurable Processor MuCCRA, *IEICE Technical Reports, RECONF2006-30*, pp. 19-24, September 2006. (In Japanese)

[42] Vu Manh Tuan, Yohei Hasegawa, Naohiro Katsura, and Hideharu Amano. Performance Evaluation of Hardware Multi-process Execution on the Dynamically Reconfigurable Processor, *IEICE Technical Reports, RECONF2006-31*, pp. 25-30, September 2006.

[43] Takashi Nishimura, Yohei Hasegawa, and Hideharu Amano. Low Power Design Technique on Dynamically Reconfigurable Processor, *IEICE Technical Reports, RECONF2006-7*, pp. 37-42, May 2006. (In Japanese)

[44] Takuro Nakamura, Hideharu Amano, Yohei Hasegawa, and Osamu Tohyama, Functional Devided Implementation of Video Application on Dynamic Reconfigurable Processor, *IEICE Technical Reports, RECONF2006-8*, pp. 43-49, May 2006. (In Japanese)

[45] Hideharu Amano, Yohei Hasegawa, and Shohei Abe, The Direct Execution Mode on Dynamically Reconfigurable Processors, *IEICE Technical Reports, RECONF2005-88*, pp. 13-18, January 2006. (In Japanese)

[46] Naohiro Katsura, Yohei Hasegawa, Vu Manh Tuan, and Hideharu Amano. Implementation of Stream Application on Programmable Devices by C Level Design, *IEICE Technical Reports, RECONF2005-82*, pp. 31-36, January 2006. (In Japanese)

[47] Shohei Abe, Yohei Hasegawa, Takao Toi, Takeshi Inuo, and Hideharu Amano. Adaptive Computing on the Dynamically Reconfigurable Processor DRP-1, *IEICE Technical Reports, RECONF2005-57*, pp. 25-30, November 2005. (In Japanese)

[48] Shohei Abe, Yohei Hasegawa, Syunsuke Kurotaki, Kenichiro Anjo, Toru Awashima, and Hideharu Amano. Implementation of AES-CBC on the Dynamically Reconfigurable Processor, *The 4th Reconfigurable Systems Symp.*, pp. 239-242, September 2004. (In Japanese)

[49] Hideharu Amano, Shohei Abe, Katsuaki Deguchi, Yohei Hasegawa, Input/Output mechanisms of dynamically reconfigurable processors -Move data or change configuration-, *IEICE Technical Reports, CPSY2004-40*, pp. 47-52, December 2004. (In Japanese)

Appendix A

MuCCRA-1 Configuration Specification

A.1 Configuration Maps

This section describes the configuration maps for each reconfigurable element of the MuCCRA-1.

A.1.1 PE Configuration Map

The configuration map for PE is shown in Fig. A.1. A context of a PE is represented by a 64-bit configuration data; 48bits for the PE Core (PECONF) and 16bits for the PICKIN and PICKOUT (PICKIN/PICKOUT).

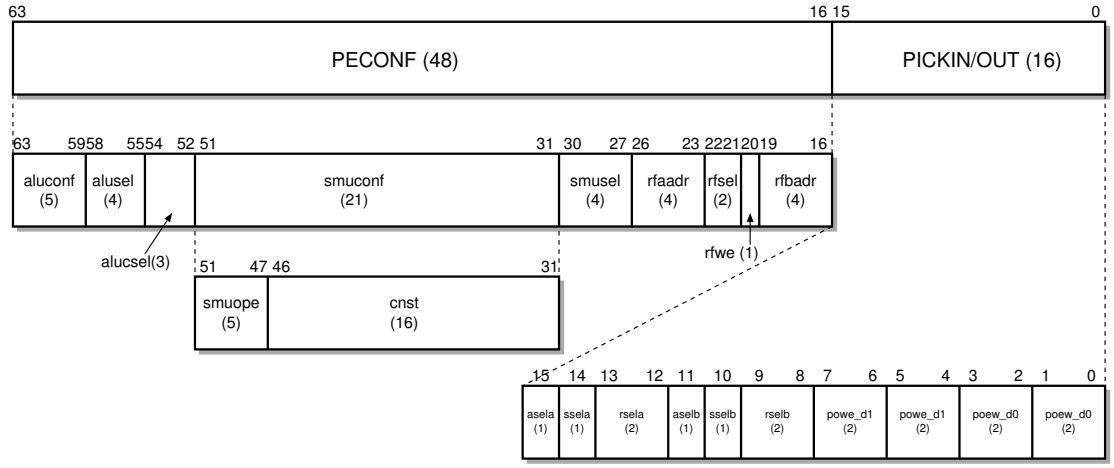


Fig. A.1: PE Configuration Map (PECONF)

The PECONF includes the following entries:

- aluconf: an instruction opecode for the ALU,
- alusel: select signals for two inputs of the ALU (A and B),
- alucsel: select signal for a carry input of the ALU (C),
- smuope: an instruction opecode for the SMU,

- cnst: a constant value for the SMU,
- smusel: select signals for two inputs of the SMU (A and B),
- rfaddr: a read/write address for port A of the RFile,
- rfsel: select signals for two inputs of the RFile (rfina and rfinc),
- rfsel: a write enable signal for the RFile, and
- rfbddr: a read address for port B of the RFile.

For the configuration data (PICKIN/PICKOUT), asela, ssela, and rsela mean select signals of the left-side PICKIN block for inputs of ALU, SMU, and RFile, respectively. Similarly, aselb, sselb, and rselb are for the right-side PICKIN block. In addition, powea, powed,poewa, and poewd are select signals for the PICKOUT blocks for each direction (East/West) and each channel (d0/d1).

A.1.2 SE Configuration Map

The configuration map for SE is shown in Fig. A.2. A context of a SE is represented by a 16-bit configuration data (SECONF), and it includes the configuration data for two switches (SWs).

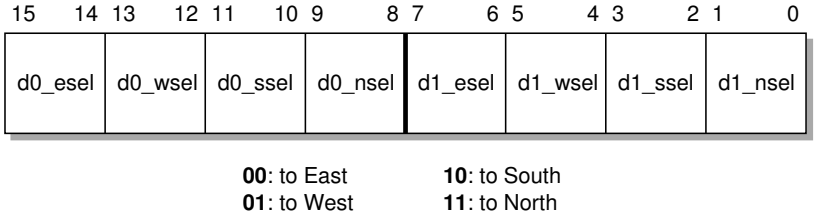


Fig. A.2: SE Configuration Map (SECONF)

The configuration data includes eight 2-bit data, and each datum indicates the switching data for each destination direction (East/West/South/North) and each channel (d0/d1). For example, d0_esel means the select signal of the switch to the East direction for the channel d0.

A.1.3 MEM Configuration Map

The configuration map for MEM is shown in Fig. A.3. A context of a MEM is represented by a 9-bit configuration data (MEMCONF).

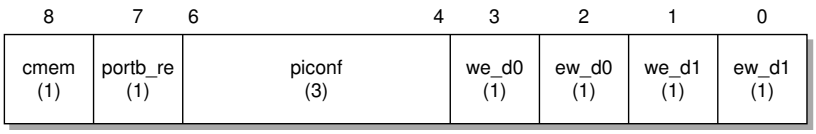


Fig. A.3: MEM Configuration Map (MEMCONF)

The PECONF includes the following entries:

- cmem: A user-defined write enable signal for all ports of the MEM,
- portb_re: A read enable signal for port B of the MEM,
- piconf: A select signal for the PICKIN block of the MEM, and
- we_d0, ew_d0, we_d1, ew_d1: Select signals for the PICKIOUT blocks of the MEM.

A.1.4 MULT Configuration Map

The configuration map for MULT is shown in Fig. A.4. A context of a MULT is represented by a 16-bit configuration data (MULTCONF), and it includes the select signals of PICKIN blocks for all the dedicated multipliers.

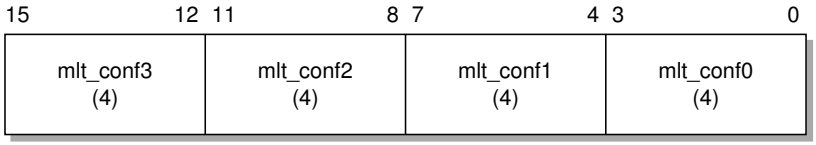


Fig. A.4: MULT Configuration Map (MULTCONF)

A.1.5 CSC Configuration Map

The configuration map for CSC is shown in Fig. A.5. A context of a CSC is represented by a 10-bit configuration data (CSCCONF).

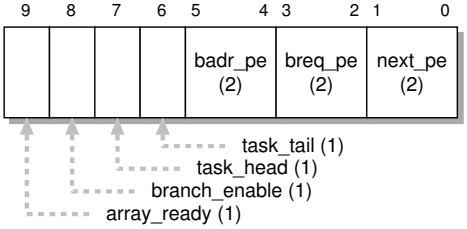


Fig. A.5: CSC Configuration Map (CSCCONF)

The (PECONF) includes the following control signals:

- array_ready: A ready signal for starting the IO transfer,
- branch_enable: A context branch enable signal,
- task_head: A flag for specifying if the context is a head of the task,
- task_tail: A flag for specifying if the context is a tail of the task,
- badr_pe: A select signal for specifying the selected special PE which generates a branch address (badr) in the context,
- breq_pe: A select signal for specifying the selected special PE which generates a task branch request signal (breq) in the context,

- next_pe: A select signal for specifying the selected special PE which generates a task branch enable signal in the context,

A.2 Instruction Sets

A.2.1 ALU Instruction Set

The instruction set for an ALU of the MuCCRA-1 is shown in Table A.1. In addition to the basic 19 instructions including selections, logical operations (AND/NAND/OR/NOR/EXOR), addition, subtractions, and comparisons, the half-ward instructions are supported for each basic instruction. Furthermore, it also has the packing and unpacking instructions for an efficient data handling.

A.2.2 SMU Instruction Set

The instruction set for an SMU of the MuCCRA-1 is shown in Table A.2. Each SMU supports the basic shift and mask operations including logical and arithmetic shifts. In addition, the operations which treat a constant value are also implemented. As similar to the ALU, the SMU can perform half-word operations.

A.3 Field Map of Configuration Data

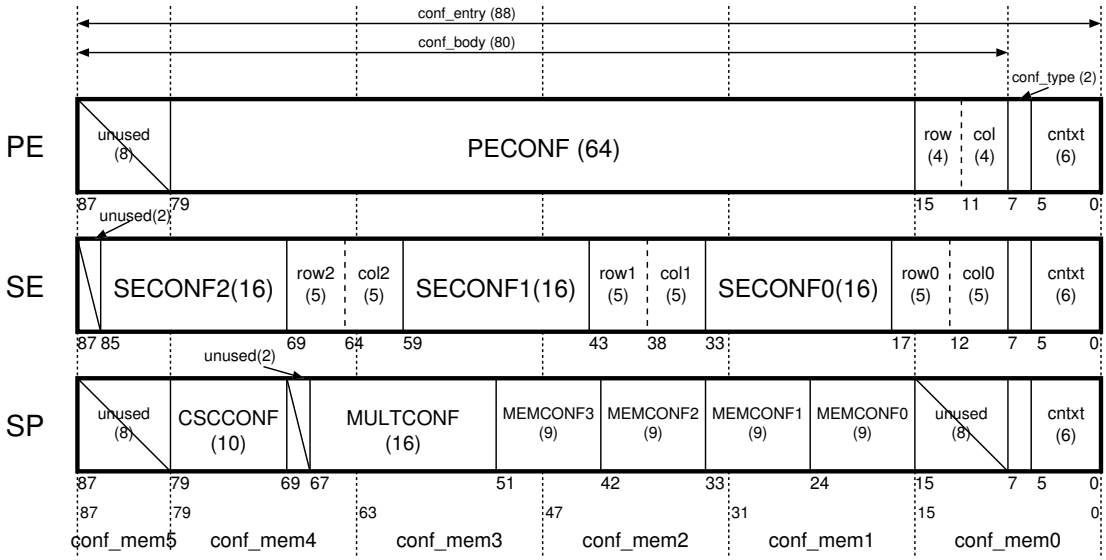


Fig. A.6: Field Map of Configuration Data

Fig. A.6 shows the address map of the configuration data memory in the TCC. The configuration data memory consists of four single-port SRAM cells (conf_mem0 to conf_mem5). And, each entry of the configuration data memory is 88-bit wide (conf_entry), and it includes multicast bits for

Table A.1: ALU Instruction Set of MuCCRA-1

aluconf	mnemonic	out	outc
00000	SELA	C1 ? A : B	C
00001	SELTA	C1 ? ~A : B	C
00010	SELB	C1 ? B : A	C
00011	SELTB	C1 ? ~B : A	C
00100	AND	A & B	C
00101	NAND	~(A & B)	C
00110	NANDTA	~(~A & B)	C
00111	NANDTB	~(A & ~B)	C
01000	OR	A | B	C
01001	NOR	~(A | B)	C
01010	NORTA	~(~A | B)	C
01011	NORTB	~(A | ~B)	C
01100	EXOR	A ^ B	C
01101	EXORTB	A ^ ~B	C
01110	ADD	A + B + C1	{carry1, C0}
01111	SUBAB	A + ~B + C1	{sign1, C0}
10000	SUBBA	~A + B + C1	{sign1, C0}
10001	EQL	A	{(A== B), C0}
10010	EQLTB	A	{(A==~B), C0}
10011	HSELA	{(C1 ? A1 : B1), (C0 ? A0 : B0)}	C
10100	HSELTA	{(C1 ? ~A1 : B1), (C0 ? ~A0 : B0)}	C
10101	HSELB	{(C1 ? B1 : A1), (C0 ? B0 : A0)}	C
10110	HSELTB	{(C1 ? ~B1 : A1), (C0 ? ~B0 : A0)}	C
10111	HADD	{(A1 + B1 + C1), (A0 + B0 + C0)}	{carry1, carry0}
11000	HSUBAB	{(A1 + ~B1 + C1), (A0 + ~B0 + C0)}	{sign1, sign0}
11001	HSUBBA	{(~A1 + B1 + C1), (~A0 + B0 + C0)}	{sign1, sign0}
11010	HEQL	A	{(A1== B1), (A0== B0)}
11011	HEQLTB	A	{(A1==~B1), (A0==~B0)}
11100	PCKH	{A1, B1}	C
11101	PCKL	{A0, B0}	C
11110	UPCKLA	A & 24'h000FFF	C
11111	UPCKLB	B & 24'h000FFF	C

A1 = A[23:12], A0 = A[11:0],
B1 = B[23:12], B0 = B[11:0],
C1 = C[1], C0 = C[0]

Table A.2: SMU Instruction Set of MuCCRA-1

smuope	mnemonic	out	outc
00000	SLI_AND	(A << cnst[4:0]) & B	out[23:22]
00001	SLI_OR	(A << cnst[4:0]) | B	out[23:22]
00010	SLI_EOR	(A << cnst[4:0]) ^ B	out[23:22]
01000	SRI_AND	(A >> cnst[4:0]) & B	out[1:0]
01001	SRI_OR	(A >> cnst[4:0]) | B	out[1:0]
01010	SRI_EOR	(A >> cnst[4:0]) ^ B	out[1:0]
00100	SHL_ANDI	(A << B[4:0]) & cnst24	out[23:22]
00101	SHL_ORI	(A << B[4:0]) | cnst24	out[23:22]
00110	SHL_EORI	(A << B[4:0]) ^ cnst24	out[23:22]
01100	SHR_ANDI	(A >> B[4:0]) & cnst24	out[1:0]
01101	SHR_ORI	(A >> B[4:0]) | cnst24	out[1:0]
01110	SHR_EORI	(A >> B[4:0]) ^ cnst24	out[1:0]
10000	SLI_ANDTB	(A << cnst[4:0]) & ~B	out[23:22]
10001	SLI_ORTB	(A << cnst[4:0]) | ~B	out[23:22]
10010	SLI_EORTB	(A << cnst[4:0]) ^ ~B	out[23:22]
10100	SLI_NOR	~(A << cnst[4:0]) & ~B	out[23:22]
10101	SLI_NAND	~(A << cnst[4:0]) | ~B	out[23:22]
01111	SRAI_AND	(A >>> cnst[4:0]) & B	out[23:22]
10011	SRAI_OR	(A >>> cnst[4:0]) | B	out[23:22]
01111	SRAI_EOR	(A >>> cnst[4:0]) ^ B	out[23:22]
00111	IMM	cnst24	out[1:0]
11000	LHI	cnst24 << 16	cnst[9:8]
00011	EQLA	A	{A==(B|cnst24),1'b0}
01011	EQLI	cnst24	{A==B, A==cnst24}
10110	HSHL	{A1 << B1[3:0], A0 << B0[3:0]}	{out[23], out[11]}
10111	HSHR	{A1 >> B1[3:0], A0 >> B0[3:0]}	{out[23], out[11]}
11110	HSLI	{A1 << cnst[11:8], A0 << cnst[3:0]}	{out[23], out[11]}
11001	PCKH	{A1, B1}	2'b10
11010	PCKL	{A0, B0}	2'b01
11011	ROLI	(A << cnst[4:0])|(A >> (24-cnst[4:0]))	out[23:22]
11100	ROL	(A << B[4:0]) | (A >> (24-B[4:0]))	cnst[1:0]
11101	ROR	(A >> B[4:0]) | (A << (24-B[4:0]))	cnst[1:0]

The 24-bit constant `cnst24` is sign extension of 16-bit constant `cnst`.

```
A1 = A[23:12], A0 = A[11:0],
B1 = B[23:12], B0 = B[11:0]
```

the RoMultiC and the context number in addition to the configuration data for each reconfigurable element. An entry of the configuration data memory has a 2-bit type of the configuration data called conf_type, and it is either of PE, SE, and SP. The conf_type specifies the type of reconfigurable elements which can load the configuration data.

A.4 MuCCRA-1 I/O Address Map

The MuCCRA-1 has 64-bit wide input and output ports, respectively. For the data input and output, the distributed memory modules (MEMs) are used as an I/O interface, and they are double-buffered in order to overlap the I/O and computation as described in Section 6.3.4. The input port is also used to input configuration data into the configuration data memory in the TCC.

Fig. A.7 describes the I/O address map of the MuCCRA-1. On the 12-bit wide address space, MEMs, 8-entry TFT register, and configuration data memory modules are mapped. Each of the MEMs is composed of a 16-bit × 512-word SRAM cell and an 8-bit × 512-word SRAM cell. The TFT register is an 8-entry register file, and it implies the TCC can manage up to 8 independent tasks as shown in Section 6.3.2. The configuration data memory in the TCC is composed of five 16-bit × 512-word SRAM cells and an 8-bit × 512-word SRAM cell.

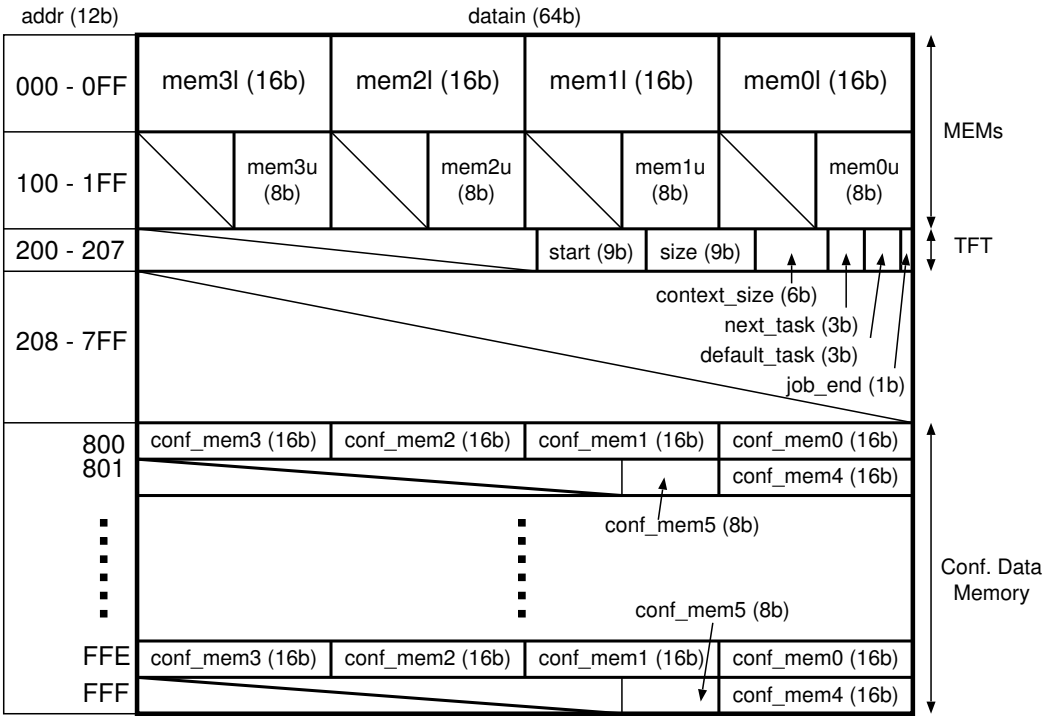


Fig. A.7: MuCCRA-1 Address Map