Adaptive Computing based on FPGA Run-time Reconfigurability

Ming Liu

Stockholm 2011

Thesis submitted to the Royal Institute of Technology in partial fulfillment of the requirements for the degree of Doctor of Technology

Liu, Ming
Adaptive Computing based on FPGA Run-time Reconfigurability

ISBN 978-91-7415-985-1
TRITA-ICT/ECS AVH 11:05
ISSN 1653-6363
ISRN KTH/ICT/ECS/AVH-11/05-SE

Royal Institute of Technology
School of Information and Communication Technology
Department of Electronic Systems
Forum 120
SE-164 40 Kista, Sweden

Abstract

In the past two decades, FPGA has been witnessed from its restricted use as glue logic towards real System-on-Chip (SoC) platforms. Profiting from the great development on semiconductor and IC technologies, the programmability of FPGAs enables themselves wide adoption in all kinds of aspects of embedded designs. Modern FPGAs provide the additional capability of being dynamically and partially reconfigured during the system run-time. The run-time reconfigurability enhances FPGA designs from the sole spatial to both spatial and temporal parallelism, providing more design flexibility for advanced system features.

Adaptive computing delegates an advanced computing paradigm in which computation tasks and resources are intelligently managed in correspondence with conditional requirements. In this thesis, we investigate adaptive designs on FPGA platforms: We present a comprehensive and practical design framework for adaptive computing based on the FPGA run-time reconfigurability. It concerns several design key issues in different hardware/software layers, specifically hardware architecture, run-time reconfiguration technical support, OS and device drivers, hardware process scheduler, context switching as well as Inter-Process Communications (IPC). Targeting a special application of data acquisition (DAQ) and trigger systems in nuclear and particle physics experiments, we set up the data streaming model and conduct theoretical analysis on the adaptive system. Three application studies are employed to verify the proposed adaptive design framework: The first application demonstrates a peripheral controller adaptable system aiming at general embedded designs. Through dynamically loading/unloading a NOR flash memory controller and an SRAM controller, both flash memory and SRAM accesses may be accomplished with less resource consumption than in traditional static designs. In the second case, two real algorithm processing engines are adaptively time-multiplexed in the same reconfigurable slot for particle recognition computation. Experimental results reveal the reduced on-chip resource requirements, as well as

an approximate processing capability of the peer static design. Taking advantage of the FPGA dynamic reconfigurability, we present in the third application a novel on-FPGA interconnection microarchitecture named RouterLess NoC (RL-NoC). RL-NoC employs the novel design concept of Move Logic Not Data (MLND), and significantly distinguishes itself from the existing interconnection architectures such as buses, crossbars or NoCs. It does not rely on routers to distribute packets as canonical NoCs do, but time-share point-to-point communication channels among different source-destination node pairs with packet injection and retrieval decoupled. In comparison with canonical packet-switching NoCs, the routerless architecture features lower design complexity, less resource consumption, higher work frequency, more efficient power dissipation as well as comparable packet delivery efficiency. It is regarded as a promising interconnection approach in some design scenarios on FPGAs, especially for light-weight applications.

Acknowledgments

This thesis project has been carried out under the collaboration between Department of Electronic Systems of Royal Institute of Technology (KTH) in Stockholm, Sweden, and II. Physics Institute of Justus-Liebig-University (JLU) in Giessen, Germany. It was supported in part by BMBF under contract Nos. 06GI9107I and 06GI9108I, FZ-Jülich under contract No. COSY-099 41821475, HIC for FAIR, and WTZ: CHN 06/20.

Most of all, I would like to thank my three supervisors, respectively Professor Wolfgang Kühn, Dr. Zhonghai Lu, and Professor Axel Jantsch for providing me the opportunity to do such an interesting interdisciplinary work. Professor Kühn is my local supervisor in Germany. I learned from him different cultures in physics and also different methods to solve problems. His broad knowledge in physics, computer and electronics areas impresses me very much. I thank him also for his enlightenment in our conversation, especially when I encountered spiritual or technical obstacles in my study and work. Dr. Lu is my direct supervisor who gave me most instructions. He devoted quite much private time on my study. I give my greatest appreciation to him for his fruitful advice on the process of problem targeting, solution proposal, experimental setup and scientific writing. Last but not the least, Professor Jantsch is a very respectable person for his personality, knowledge and inspiration to students. I still clearly remember his words when I was enrolled as a Ph.D student: "Ph.D study is exactly like the sailing Columbus on the sea. You can never know which new land you will arrive at unless you go ahead and try to search for."

I am also thankful to all my colleagues in Giessen and Stockholm. The discussion and suggestion from them are so helpful to improve my professional knowledge and technical skills. My current and previous Giessen colleagues, specifically Jens Sören Lange, Vladimir Pechenov, Geydar Agakishiev, Olga Pechenova, Marco Destefanis, Stefano Spataro, Daniel Kirschner, Johannes Roskoss, Camilla Kirchhübel, Andreas Kopp, Johannes Lang, Zoltán Gagyi-

Pálfy, Thomas Gessler, David Münchow, Ingo Heller, Matthias Ullrich, Marcel Werner, Martin Galuska, Stephanie Künze, Sören Fleischer, Björn Spruck, Yutie Liang, Li Lu, and so on, explained me plenty of physics background knowledge which makes me understand the application very clearly. I also thank Ingo Sander, Johnny Öberg, and Vladimir Vlassov for their interesting lectures and discussion on the modern techniques, as well as Huimin She, Jun Zhu, Liang Rong, Geng Yang, Xiaolong Yuan, Zhuo Zou, Peng Wang and Jiayi Zhang for exchanging our experience of Ph.D study. Many thanks are given to our Chinese collaboration group in Beijing, including Zhen'an Liu, Hao Xu, Qiang Wang, Dapeng Jin for their nice work on the compute node PCB design.

I appreciate Christa Momberger, Thomas Köster, Lena Beronius, Agneta Herling and Alina Munteanu for their administrative and non-technical assistance in travel arrangements, device ordering, and many other issues.

Many thanks to all our international collaboration group members in Munich, Darmstadt, Jülich, Cracow, Uppsala, and other cities. Due to the large quantity I cannot list all the names here. I appreciate very much for our delightful technical discussion in collaboration meetings.

Greatest thanks to my parents and relatives who are far away in China. Their continuous encouragement and support gave me strength to overcome any kind of difficulties in all aspects. I want to tell them loudly, Mama and Papa, thank you for your endless support on my study. I love you!

Special thanks go to my girlfriend Shuo who accompanies me through the life in Europe. I will forever remember the days we spent together, busily but happily.

Ming Liu

April 2011, Giessen

Contents

List of Figures

List of Tables

Abbreviations

AHB	AMBA High-performance Bus
ALICE	A Large Ion Collider Experiment
AMBA	Advanced Microcontroller Bus Architecture
ARM	Advanced RISC Machine
ASIC	Application Specific Integrated Circuit
ATCA	Advanced Telecommunications Computing Architecture
ATLAS	A Toroidal LHC ApparatuS
BESIII	BEijing Spectrometer 3
BM	Bus Macro
BRAM	Block RAM
CAMAC	Computer-Aided Measurement And Control
CMS	Compact Muon Spectrometer
CN	Compute Node
CPU	Central Processing Unit

DAQ	Data AcQuisition
DDR	Double Data Rate
DMA	Direct Memory Access
DSP	Digital Signal Processor
FIFO	First-In-First-Out
FPGA	Field Programmable Gate Array
FRU	Field Replaceable Unit
FSM	Finit State Machine
GPCPU	General-Purpose Central Processing Unit
GPIO	General-Purpose Input/Output
HADES	High Acceptance Di-Electron Spectrometer
HDL	Hardware Description Language
HW	Hardware
IC	Integrated Circuit
ICAP	Internal Configuration Access Port
IP	Intelligent Property
IPC	Inter-Process Communication
IPMC	Intelligent Platform Management Controller
ISE	Integrated Software Environment
LE	Logic Element

LHC	the Large Hadron Collider
LHCb	the Large Hadron Collider beauty
LUT	Look-Up Table
MDC	Mini Drift Chamber
MGT	Multi-Gigabit Transceiver
MLND	Move Logic Not Data
MOPS	Mega Operations
MPMC	Multi-Port Memory Controller
NI	Network Interface
NoC	Network-on-Chip
OPB	On-chip Peripheral Bus
OS	Operating System
P2P	Point-to-Point
PANDA	antiProton ANnihilations at DArmstadt
PC	Personal Computer or Physical Channel
PCB	Printed Circuit Board
PIM	Personality Interface Modules
PLB	Processor Local Bus
PLD	Programmable Logic Device
PRM	Partially Reconfigurable Module
PRR	Partially Reconfigurable Region

QoS	Quality-of-Service
RICH	Ring Imaging CHerenkov
RL-NoC	RouterLess Network-on-Chip
RNI	Resource Network Interface
RRU	Ring Recognition Unit
SIMD	Single Instruction Multiple Data
SDF	Synchronous Data Flow
SDRAM	Synchronous Dynamic Random Access Memory
SRAM	Static Random Access Memory
SW	Software
TDM	Time-Division Multiplexing
TOF	Time-Of-Flight
TPU	Tracking Processing Unit
VC	Virtual Channel
VCF	Virtual ConFiguration
VHDL	VHSIC hardware description language
VMEbus	Versa Modular Eurocard bus
WH-NoC	WormHole Network-on-Chip
XPS	Xilinx Platform Studio

"Science is organized knowledge. Wisdom is organized life."

Immanuel Kant
(German Philosopher, 1724 - 1804 A.D.)

Chapter 1

Introduction

In this chapter we briefly introduce the technical background on FPGAs and reconfigurable computing. Drawbacks of using the conventional static approach to develop FPGAs are discussed. Motivated to improve the development methodology, we propose a promising solution of employing FPGA run-time reconfigurability to construct adaptive systems. In the end of the chapter, contributions of the author are clearly summarized with professional publications listed.

1.1 FPGA and Reconfigurable Computing

A Field-Programmable Gate Array (FPGA) is a special Integrated Circuit (IC) designed to be configurable by designers after its manufacturing, in order to obtain customized logical or arithmetic functions. It was firstly introduced by Xilinx into the commercial IC market in 1985. As the performance gap between FPGAs and ASICs decreases [1], FPGAs are playing a more and more important role in modern embedded designs. Modern FPGA products take the advantage of semiconductor technology advances, combining both programmable Logic Elements (LE) and hardcore IP blocks on the same die. They provide sufficient chip capacity and design convenience to implement a real System-on-an-FPGA, which contains microprocessors, memory controllers, hardware accelerators as well as peripheral controllers, etc. Figure 1.1 shows the capacity increment of the flagship products from the two market-dominating FPGA vendors (Xilinx and Altera) in the period from 1998

to 2009. We observe that the quantity of on-chip LEs has increased by roughly 40X, from ∼20K using 220 nm process in 1998 to ∼800K using 40 nm process in 2009. This chip density increment mirrors very well the famous prediction of Moore's Law.

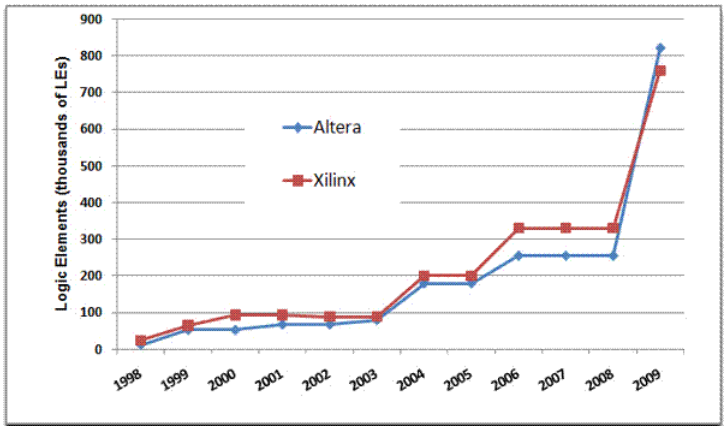


Figure 1.1. The largest FPGA announced by Xilinx and Altera (by equivalent 4-input LEs) [2]

Programmable Logic Devices (PLD) especially FPGAs are widely employed in various application fields. Figure 1.2 demonstrates the statistics of some most significant fields in the third quarter of 2009. We see the field of communications takes the largest market share of 42%. The second largest application is industrial devices. Although contributing only a very small proportion to the market share and not even shown in the figure, scientific computing benefits also largely from FPGAs, and this application field deserves extra attention due to its special meaning for the entire human being. In this thesis work, we place our research in the application background of nuclear and particle physics experiments, which study the elementary constituents of matter and their interactions. The application background will be introduced in the next chapter.

Reconfigurable computing combines both the flexibility of software programming and the high performance of hardware acceleration by using programmable computing fabrics mainly FPGAs. The principal difference in comparison with the ordinary microprocessor computing, is the capability to make substantial changes to the datapath in addition to the control flow. Actually

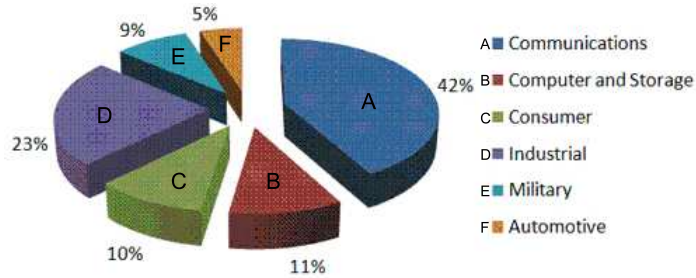


Figure 1.2. PLD market by end applications in the third quarter of 2009

the basic concept can be traced back to the 1960s, when Gerald Estrin's landmark paper proposed the concept of a computer made of a standard processor and an array of "reconfigurable" hardware [3] [4]. The main processor controls the behavior of the reconfigurable hardware. The latter can be tailored to perform a specific task as quickly as a dedicated piece of hardware. Unfortunately this idea was far ahead of its time in terms of needed electronics technologies. Only after the great development of configurable devices and corresponding EDA tools in the recent decade, reconfigurable computing could be really widely adopted to achieve performance benefits as well as flexible reprogrammability.

Although normally running at a much lower clock frequency, FPGA-based reconfigurable computing is believed to have a 10 - 100X accelerated performance but far lower power consumption compared to microprocessors. The development methodology on the processor software focuses on flexible control flows with limited number of Processing Elements (PE), while more concerns are placed on the datapath design and optimization through parallel and pipelined approaches when developing reconfigurable hardware platforms. The on-chip memory concurrency and fine-grained computation parallelism may overcome the bottleneck existing in the computer memory system. More efficiently utilizing the expensive chip area, reconfigurable computing creates an unprecedented opportunity for orders of magnitude improvement in MOPS/dollar, MOPS/watt, and just MOPS.

In our target physics experiment applications, FPGA-based reconfigurable solutions have significant advantages to implement application-specific algorithms. They have comparatively simple control flows during data processing, and optimized datapath designs can result in high performance with the on-

chip memory concurrency and fine-grained computation parallelism or pipeline support. The reprogrammability enables to change the designs to meet different experimental requirements. In addition, the ASIC production start-up cost can be saved, considering the required comparatively small quantity of chips in experimental facilities.

1.2 FPGA-based System Examples

Reconfigurable computing satisfies the simultaneous demands of application performance and flexibility. In the present era when cluster-based supercomputers still dominate the fields of super computation tasks, reconfigurable computing begins showing large potential and perspective on many performance-critical areas such as realtime scientific computing. Currently commercial and academic projects are developing hardware and software to employ the raw computational power of FPGAs. Among them some platforms are augmented computer clusters with FPGAs attached to the system bus as accelerator blocks. One commercial example is the products from Cray Inc., such as XD1, XT5h series supercomputers. FPGAs are integrated in the system to embody various digital hardware designs and augment the processing capability of AMD Opteron processors [5] [6] [7]. Another instance in physics experiment applications is the ATLAS level 2 trigger [8]. Their design appears as PCI cards in commodity PCs, in which only those simple but computing-intensive steps are released to FPGAs while others remain on CPUs. One major weakness of these systems is the bandwidth bottleneck between the host microprocessor and the FPGA accelerator. In the ATLAS case, the computation work relies much on the PC and the limited bandwidth between CPUs and FPGAs via the PCI bus becomes the bottleneck when partitioning the algorithm steps to the CPU and the FPGA. There exist meanwhile standalone platforms in which FPGA units are independent on the host processors of commodity PCs. Usually in the FPGA design, less powerful embedded hardcore or softcore microprocessors are incorporated in the system, as the alternative of PC processors to conduct controls or HW/SW coprocessing tasks. For instance the Dini Group products [9] combine multiple FPGA chips through on-board interconnections and target mainly logic emulation and ASIC verification applications. In addition the Berkeley Emulation Engine (BEE) represents powerful and scalable platforms for large scale data processing [10] [11] [12]. Its various product generations employ different communication standards, such as Infiniband, Gigabit Ethernet, or PCI Express, etc., to network the Printed Circuit Boards (PCB) for system scaling. The

recent COPACOBANA [13] and the CUBE [14] projects use plentiful interconnected FPGAs to run cryptanalytical algorithms. Specific to their application requirements, the hardware design suits to solve parallel computation-hungry problems partitioned into many FPGA chips, but features small memory bandwidth and capacity as well as low inter-chip communication requirements.

According to the computation and communication requirements in our target physics applications, we have designed a hierarchical and scalable FPGA-based computation platform that is optimized to interface with other experimental devices and conducts the large-scale data processing work. Details on the hardware and FPGA development may be referred to in Appendix A.

1.3 Motivation

Despite large advances on the density and work frequency, the chip area utilization efficiency and the system clock speed of FPGAs are still low compared to the gate-based logic implementation on ASICs. On the FPGA, Look-Up Table (LUT) units are employed instead to construct combinational logic, leading to larger area occupation on the die and slower work frequency than elementary gates. In [15], the authors have measured FPGA designs to be 35X larger in area and 3X slower in speed than a standard cell ASIC flow, both using 90-nm technology. In [16], a 12 year old Pentium design was ported on a Xilinx Virtex-4 FPGA. A 3X slower system speed (25 MHz vs. 75 MHz) is still observed, even though the FPGA uses a recent 90-nm technology while the original ASICs were 600-nm. The speed and area utilization gap between FPGAs and ASICs has also been differently quantified in [17] and [18] for various designs. We see that FPGA programmable resources are still comparatively expensive. Efficient resource management and utilization remain to be a challenge especially for the applications with simultaneous high-performance and low cost requirements.

From another point of view, design complexity especially in massive processing systems is increasing rapidly and becoming unmanageable. Imagining an FPGA cluster consisting of hundreds of interconnected FPGAs, heterogeneous functional modules have to be selected and integrated in system designs. Bitstream files must be downloaded into the correct FPGA chip. The design management work is annoying and error-prone if it is done by mentally overloaded designers.

The conventional approach to develop FPGAs is static: The entire system is partitioned into tasks and implemented as hardware components. Design components are managed and assembled by designers during their development

period. They are statically mapped on the FPGA fabric, being instantiated throughout the system operation time. The functionalities normally do not change when the design is still in operation. Drawbacks exist in the static development approach, as itemized as follows:

1. **Design management complexity.** Designs are managed by designers offline. The human being must face the complexity and error possibilities, especially in heterogeneous massive systems consisting of large quantity of FPGAs.

2. **Resource utilization inefficiency.** With respect to some design modules which may only occasionally function or do not function simultaneously at all, it is a large waste to respectively allocate them on-chip resources. More utilized resources not only lead to higher static power consumption, but also may result in extra hardware upgrade budget if new functionalities are expected but the chip is already fully occupied.

3. **Incapability of run-time maintenance.** Static designs do not possess the capability of online maintenance. In order to modify or upgrade the firmware, the entire system may have to be completely stopped. This might not be permitted at all or very expensive in some situations, such as in the physics experiments.

As a promising solution, adaptive computing [19] [20] is the paradigm in which computation tasks may vary and adapt to system status or ambient conditions during run-time. Its self-awareness distinguishes itself from existing computational models, which are mostly procedural and simply a collection of static functional components. Typically an adaptive system keeps aware the context and changes its processing behavior according to trigger events such as workload variations, computation interest switching, or environmental situations. As a consequence, benefits including more efficient utilization of computation resources, lower power consumption or multitasking on limited resources may be obtained by dynamically modifying the system design or adjusting important parameters. One major prerequisite of adaptive computing, is the reprogrammability of the computer systems: On General-Purpose microprocessors (GPCPU), different computation tasks can be easily accomplished by conditionally branching to different instructions. Nevertheless hardware logic is not straightforwardly adaptable in contrast to the software computation, in which the computation resource of CPU cores is intrinsically time-shared among various tasks. Thanks to the newly emerged Partial Reconfiguration (PR) technology, which offers the possibility to dynamically change part of the

FPGA design without stopping the remaining system. The PR feature provides much convenience in realizing adaptive computing scenarios, in which basic functions are to be maintained while specific algorithms or algorithm steps can be freely adjusted. It is the PR technology that firstly introduces the concept of Time-Division Multiplexing (TDM) into FPGA resource managements. It leads to the benefits of more efficient resource utilization, shorter reconfiguration time, as well as lower static power dissipation [21] [22] [23].

1.4 Thesis Outline and Author's Contributions

The thesis is constructed into chapters listed as follows:

- Chapter 1: In this chapter we recall the readers with related technical background on FPGAs and reconfigurable computing. We analyze the main-stream development approach in existence and raise the necessity to improve the design methodology with self-adaptation features. In the end of the chapter, relevant publications on which this thesis is based are listed.

- Chapter 2: Since we target a very special application in high-energy physics experiments, it is necessary to reveal a picture of their particularities to the readers who are not familiar with this interdisciplinary field. Hence we do a brief application introduction in this chapter, and the knowledge on the application background also helps to understand the motivation of the research work.

- Chapter 3: In this chapter we present the proposed design framework for adaptive computing based on FPGA run-time reconfigurability. The framework is comprehensive and key issues are systematically discussed in different hardware/software layers. These aspects should be taken into account when practically building an adaptive design on the FPGA.

- Chapter 4: Right after the discussion on the design framework, we employ a case study to verify the concept: Two types of memory controllers time-share the same reconfigurable slot and they are adaptively loaded to work according to different memory addressing requirements. This case study reveals general benefits for FPGA-based embedded designs, on the basis of accomplishing all the expected functionalities in the system.

- Chapter 5: This chapter focuses on the performance analysis of adaptive multi-stream processing. Correlated multi-streaming acts as the

fundamental model of our target applications in high-energy physics experiments. Theoretical analysis and experimental results on the model show the potential merits of applying the self-adaptation concept in data stream processing scenarios.

- Chapter 6: Combined with the model analysis in the previous chapter, we apply the adaptively reconfigurable framework to the case study of a real application for particle recognition computation. Two pattern recognition algorithms are alternately configured to process their respective data streams, with their synchronization (correlation) requirement automatically and optimally guaranteed. Experimental results demonstrate practical design benefits.

- Chapter 7: Taking advantage of the run-time reconfigurability, we introduce a novel on-FPGA interconnection microarchitecture called Router-Less Network-on-Chip (RL-NoC) in this chapter. From the principle point of view, it significantly distinguishes itself from the conventional interconnection approaches such as buses, crossbars or NoCs. RL-NoC features many advantages in the aspects of design complexity, resource utilization, power consumption, operation frequency, etc. Meanwhile a comparable or even superior throughput performance can be achieved in comparison with a canonical wormhole NoC.

- Chapter 8: In this chapter we conclude the thesis. We summarize the advantages of the discussed design framework for adaptive computing as application references, and raise open issues for the future work in this field.

- Appendix A: In this appendix chapter, related engineering work is concerned on the FPGA-based reconfigurable platform design. This large-scale system aims at our physics applications of data acquisition (DAQ) and triggering in experiments. Moreover, the FPGA board works as the experiment platform on which researches have been carried out.

- Appendix B: The application-specific computation is introduced in this appendix chapter, specifically MDC particle track reconstruction and RICH Cherenkov ring recognition. This part consists of physics background in the detector system, algorithm description, as well as hardware implementations on FPGAs. The hardware implementation of both algorithms behaves as the basis of the second case study discussed in Chapter 6.

The author's publications are classified into three areas: system design and development, algorithm implementation, and adaptive computing. The last topic represents the author's main research contribution and it runs throughout the content of this thesis. The former two topics are application related and discussed in the appendices. The publications are itemized as follows:

Adaptive computing:

1. **Ming Liu**, Zhonghai Lu, Wolfgang Kuehn, and Axel Jantsch, "A Hardware/Software Design Framework for FPGA-based Self-aware Adaptive Computing", under submission. [24]

2. **Ming Liu**, Zhonghai Lu, Wolfgang Kuehn, and Axel Jantsch, "A Lightweight Routerless Network-on-Chip Infrastructure using FPGA Dynamic Reconfigurability", under submission. [25]

3. **Ming Liu**, Zhonghai Lu, Wolfgang Kuehn, and Axel Jantsch, "Adaptively Reconfigurable Controller for the Flash Memory", Book of *Flash Memory*, invited book chapter, InTech, ISBN: 978-953-307-272-2, 2011. [26]

4. **Ming Liu**, Zhonghai Lu, Wolfgang Kuehn, and Axel Jantsch, "Inter-Process Communications using Pipes in FPGA-based Adaptive Computing", *In Proceeding of the IEEE Computer Society Annual Symposium on VLSI*, Lixouri Kefalonia, Greece, Jul. 2010. [27]

5. **Ming Liu**, Zhonghai Lu, Wolfgang Kuehn, and Axel Jantsch, "Reducing FPGA Reconfiguration Time Overhead using Virtual Configurations", *In Proceeding of the International Workshop on Reconfigurable Communication Centric System-on-Chips*, Karlsruhe, Germany, May 2010. [28]

6. **Ming Liu**, Zhonghai Lu, Wolfgang Kuehn, and Axel Jantsch, "FPGA-based Adaptive Computing Architecture for Correlated Multi-stream Processing", *In Proceeding of the Design, Automation and Test in Europe conference*, Dresden, Germany, Mar. 2010. [29]

7. **Ming Liu**, Zhonghai Lu, Wolfgang Kuehn, Shuo Yang and Axel Jantsch, "A Reconfigurable Design Framework for FPGA Adaptive Computing", *In Proceeding of the International Conference on ReConFigurable Computing and FPGAs*, Cancun, Mexico, Dec. 2009. [30]

8. **Ming Liu**, Wolfgang Kuehn, Zhonghai Lu, and Axel Jantsch, "Run-time Partial Reconfiguration Speed Investigation and Architectural Design

Space Exploration", *In Proceeding of the International Conference on Field Programmable Logic and Applications*, Prague, Czech Republic, Aug. 2009. [31]

System design and development:

9. **Ming Liu**, Wolfgang Kuehn, Soeren Lange, Shuo Yang, Johannes Roskoss, Zhonghai Lu, Axel Jantsch, Qiang Wang, Hao Xu, Dapeng Jin, and Zhenan Liu, "A High-end Reconfigurable Computation Platform for Nuclear and Particle Physics Experiments", *Computing in Science and Engineering*, vol. 13, no. 2, pp. 52-63, Mar./Apr. 2011. [32]

10. Qiang Wang, Axel Jantsch, Dapeng Jin, Andreas Kopp, Wolfgang Kuehn, Johannes Lang, Soeren Lange, Lu Li, **Ming Liu**, Zhenan Liu, Zhonghai Lu, David Muenchow, Johannes Roskoss, and Hao Xu, "Hardware/Software Co-design of an ATCA-based Computation Platform for Data Acquisition and Triggering", *In Proceeding of the IEEE NPSS Real Time Conference*, Beijing, China, May 2009. [33]

11. **Ming Liu**, Johannes Lang, Shuo Yang, Tiago Perez, Wolfgang Kuehn, Hao Xu, Dapeng Jin, Qiang Wang, Lu Li, Zhenan Liu, Zhonghai Lu, and Axel Jantsch, "ATCA-based Computation Platform for Data Acquisition and Triggering in Particle Physics Experiments", *In Proceeding of the International Conference on Field Programmable Logic and Applications 2008*, Heidelberg, Germany, Sep. 2008. [34]

12. **Ming Liu**, Wolfgang Kuehn, Zhonghai Lu, Axel Jantsch, Shuo Yang, Tiago Perez, and Zhenan Liu, "Hardware/Software Co-design of a General-Purpose Computation Platform in Particle Physics", *In Proceeding of the IEEE International Conference on Field Programmable Technology*, Kitakyushu, Kokurakita, Japan, Dec. 2007. [35]

13. Wolfgang Kuehn, Camilla Gilardi, Daniel Kirschner, Johannes Lang, Soeren Lange, **Ming Liu**, Tiago Perez, Lars Schmitt, Dapeng Jin, Lu Li, Zhenan Liu, Yunpeng Lu, Qiang Wang, Shujun Wei, Hao Xu, Dixin Zhao, Krzysztof Korcyl, Jacek Tomasz Otwinowski, Piotr Salabura, Igor Konorov, and Alexander Mann, "FPGA - Based Compute Nodes for the PANDA Experiment at FAIR", *In Proceeding of the NPSS Real Time Conference*, Batavia, USA, Apr. 2007. [36]

14. Tiago Perez, Camilla Gilardi, **Ming Liu**, and Shuo Yang, "A FPGA-based Compute Node for the PANDA Data Acquisition and Trigger System", *In Proceeding of the International Winter Meeting on Nuclear Physics*, Bormio, Italy, Apr. 2007. [37]

Algorithm implementation:

15. **Ming Liu**, Zhonghai Lu, Wolfgang Kuehn, and Axel Jantsch, "FPGA-based Particle Recognition in the HADES Experiment", *IEEE Design and Test of Computers*, special issue on Design Methods and Tools for FPGA-Based Acceleration of Scientific Computing, Jul./Aug. 2011 (accepted). [38]

16. **Ming Liu**, Zhonghai Lu, Wolfgang Kuehn, and Axel Jantsch, "FPGA-based Cherenkov Ring Recognition in Nuclear and Particle Physics Experiments", *In Proceeding of the International Symposium on Applied Reconfigurable Computing*, Belfast, United Kingdom, Mar. 2011. [39]

17. **Ming Liu**, Axel Jantsch, Dapeng Jin, A. Kopp, Wolfgang Kuehn, J. Lang, L. Li, Soeren Lange, Zhenan Liu, Zhonghai Lu, D. Muenchow, V. Penchenov, Johannes Roskoss, S. Spataro, Qiang Wang, and Hao Xu, "Trigger Algorithm Development on FPGA-based Compute Node", *In Proceeding of the IEEE NPSS Real Time Conference*, Beijing, China, May 2009. [40]

18. **Ming Liu**, Wolfgang Kuehn, Zhonghai Lu, and Axel Jantsch, "System-on-an-FPGA Design for Real-time Particle Track Recognition and Reconstruction in Physics Experiments", *In Proceeding of the EUROMICRO Conference on Digital System Design*, Parma, Italy, Sep. 2008. [41]

Other publications which are not concerned in this thesis include:

19. Daniel Georg Kirschner, Geydar Agakishiev, **Ming Liu**, Tiago Perez, Wolfgang Kuehn, Vladimir Pechenov, and Stefano Spataro, "Level 3 Trigger Algorithm and Hardware Platform for the HADES Experiment", *Nuclear Instruments and Methods in Physics Research A*, Volume 598, Issue 2, pp. 598 - 604, 2008. [42]

20. Zhonghai Lu, **Ming Liu**, and Axel Jantsch, "Layered Switching for Networks on Chip", *In Proceeding of the Design Automation Conference*, San Diego, USA, Jun. 2007. [43]

"Thinking is more interesting than knowing,
but less interesting than looking."

Johann Wolfgang von Goethe
(German Playwright, Poet, Novelist and Dramatist, 1749 - 1832 A.D.)

Chapter 2

Application Background

This chapter concerns the necessary introduction of our target application: data acquisition and triggering in nuclear and particle physics experiments. This special application acts as a representative of large-scale computing in scientific researches, which benefit from modern semiconductor and FPGA technologies. It differs significantly from the common embedded designs such as consumer electronics or mobile devices, and therefore deserves a brief overview for the readers.

2.1 Nuclear and Particle Physics Experiments

Nuclear and particle physics is a branch of physics that studies the elementary constituents of matter and the interactions between them. It is also called high-energy physics because many elementary particles do not occur under normal circumstances in nature, but can be created and detected during energetic collisions of other particles, as is done in particle colliders. In the experiments, beam particles are accelerated to a velocity approaching to the speed of light and then collide with target particles. Modern nuclear and particle physics experiments, for example HADES [44] and PANDA [45] at GSI Germany, BESIII [46] at IHEP China, ATLAS, CMS, LHCb, ALICE at the LHC [47] at CERN Switzerland, achieve their goals by studying the emission direction, the energy, and the mass of the produced particles when the beam hits the target. In the experimental facilities, different kinds of detectors are adopted to generate raw data which are used to calculate and analyze the

characteristics of emitted particles after the collision. Figure 2.1 shows the exploded view of the HADES detector system as an example.

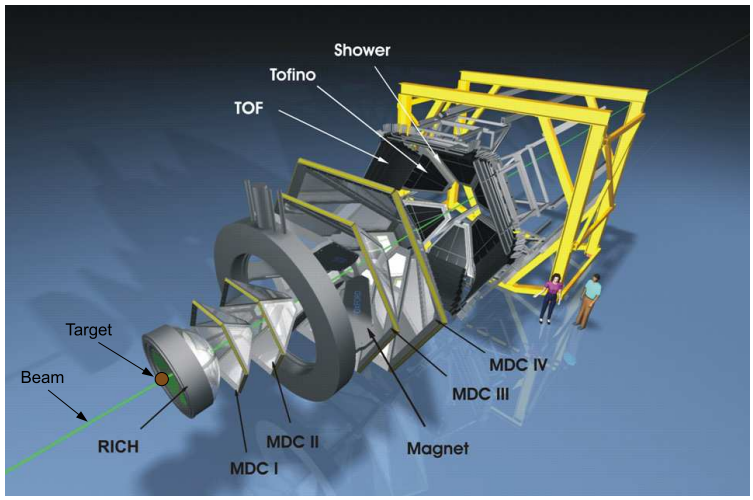


Figure 2.1. Dismounted view of the HADES detector system

In high-energy physics, one "event" corresponds to a single interaction of a beam particle with a target particle. It consists of sub-events which typically represent the information from individual detector sub-systems, such as RICH (Ring Image CHerenkov), MDCs (Mini Drift Chamber), TOF (Time-Of-Flight), etc., as shown in Figure 2.1. Figure 2.2 demonstrates an example event structure consisting of sub-events from various detectors. The event data are to be filtered and recorded during experiments. Afterwards physicists will extensively analyze the events to search for interesting ones such as new types of particles.

The detector system in experimental facilities has commonly more than 10^5 signal channels, and the delivered data rate which is the product of the event size and the reaction rate might be a scary number (e.g. PANDA, the reaction rate of 10 - 20 MHz and the data rate of more than 200 GBytes/s). Compared to the bandwidth needed by some other applications [48], Figure 2.3 lists some experiments by their event sizes and reaction rates. We see that their data rates range from 10^7 up to 10^{11} Bytes/s, which are too large for the disk or tape storage to entirely record the data throughout the experiment time lasting for months. Furthermore, the supercomputers would take forever to

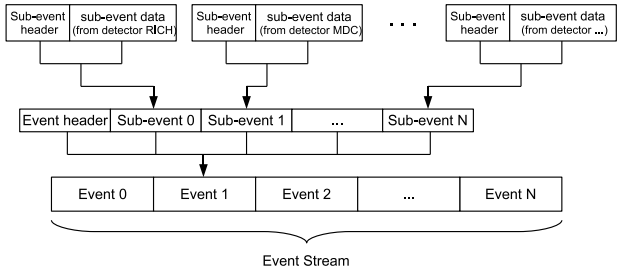


Figure 2.2. Event structure consisting of sub-events from different detectors

process all the data during offline analysis. Among the generated events, only a rare fraction with particular physics contents is of interest for the physicists. Such events might occur only once within one million interactions. Therefore it is essential to realize an efficient on-line data acquisition (DAQ) and trigger system which processes the sub-events coming from detectors and reduces the data rate by several orders of magnitude by means of rejecting the background.

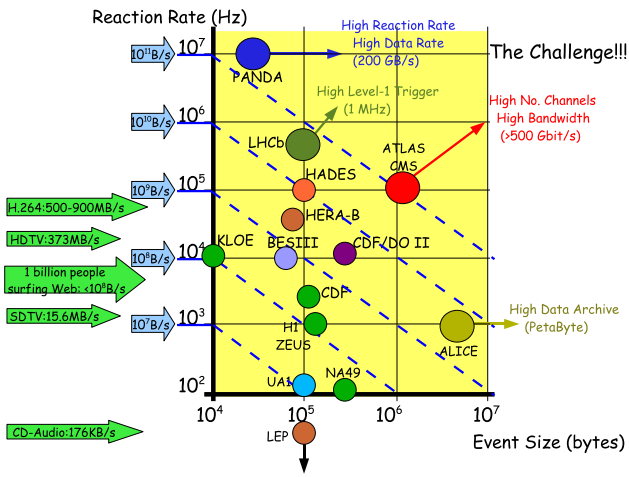


Figure 2.3. Experiments with different event sizes and reaction rates

2.2 DAQ and Trigger System

In the contemporary facilities, pattern recognition algorithms [49] [50] [51] such as *Cherenkov ring recognition*, *particle track reconstruction*, *Time-Of-Flight (TOF) processing*, *Shower recognition*, are implemented as sophisticated criteria according to detector categories. Only the sub-events which possess expected patterns generated by certain types of particles and could be successfully correlated among different detectors, trigger a positive decision and are encapsulated in a pre-defined event structure for mass storage and further offline analysis. Others will be discarded on the fly in order to reduce the data rate. Figure 2.4 demonstrates a sample multi-streaming data flow in the experiments, which will be later modeled and analyzed in Chapter 5.

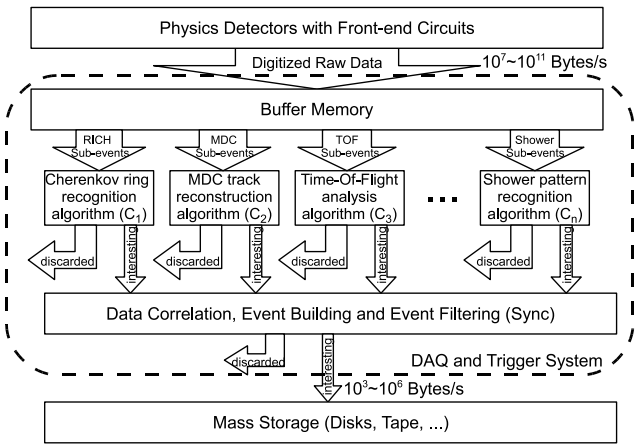


Figure 2.4. Multi-streaming data flow in DAQ and trigger systems

As an alternative approach of the traditional PC farm based pure software computation, modular approaches with commercial bus systems, such as VMEbus, FASTbus, and CAMAC, can be utilized to construct DAQ and trigger systems for high-energy physics experiments [52] [53] [54] [55]. These systems which usually contain programmable devices like FPGAs or DSPs may interface to PC clusters for hardware/software hybrid processing. An instance is the NA48 experiment [56], in which FPGAs are used to build wire hit coordinates for the pion particle (π) track reconstruction computation, and the trigger core algorithm is implemented on interconnected Digital Signal Processors (DSP). The customized hardware boards are installed on 4 VME crates.

One SUN workstation remotely monitors the entire system through a private Ethernet network. There is an additional SPARC VME SBC computer board in each crate running Unix to control the system. However due to the largely increased data rate generated by the detector systems in modern experiments, the obsolete bus-based technologies cannot meet the increasing experimental requirements any longer. The time-multiplexing nature of the system bus not only deteriorates the data exchanging efficiency among algorithms residing on different pluggable modules, but also restricts the flexibility to partition complex algorithms. Nowadays the networking and switching technologies make it efficient to construct large-scale systems for parallel and pipelined processing. In addition, great development on FPGAs provides the practicability to release some complex algorithms, which were conventionally implemented as software on workstations or embedded processors/DSPs, into the FPGA fabric for high-performance hardware processing. For example in [57], the authors utilize FPGAs to implement the Compact Muon Solenoid (CMS) trigger of the Large Hadron Collider (LHC) particle accelerator at CERN.

As a general-purpose solution for data acquisition and trigger applications in various physics experiments, specifically the HADES and BESIII upgrade and the PANDA construction projects, we have designed and developed a computation platform consisting of interconnected nodes based on the Advanced Telecommunications Computing Architecture (ATCA) standard [58]. To simultaneously satisfy various algorithm partition and correlation requirements for current or future experiments, the full-mesh Point-to-Point (P2P) backplane has been chosen to interconnect multiple FPGA-based Compute Nodes (CN) for massive parallel processing. External channels including optical links as well as Gigabit Ethernet, are engaged in receiving sub-events from detectors and forwarding processing results to the PC farm for mass storage and offline analysis. Internal hierarchical interconnections, including the inter-chassis optical link and Ethernet switching, inter-board full-mesh backplane connections, and on-board FPGA I/O channels, are utilized to partition and distribute algorithms or algorithm steps for parallel/pipelined trigger processing and correlation. In the ATCA crate, CNs appear as Field Replaceable Units (FRU), which are interconnected with each other by backplane channels. On each CN, five Xilinx Virtex-4 FX60 FPGAs are placed on board and mutually interconnected by General-Purpose I/O (GPIO) buses as well as RocketIO serial links. Four FPGAs work as algorithm processor nodes, and the fifth one as a switch interfacing to the backplane. Detailed architectural description on the ATCA-based computation platform and the CN design will be separately discussed in Appendix A. Trigger algorithm implementations on the FPGA can be found in Appendix B.

2.3 Design Challenges

Design and development of DAQ and trigger systems in high-energy physics experiments feature different requirements from other common applications, such as consumer electronics or mobile devices. Design particularities and challenges are partly summarized in the following aspects:

- **Large system scale.** The DAQ and trigger system designs typically feature an enormous scale. As we mentioned in the first section of this chapter, the raw data rate generated by particle detectors may reach up to several hundreds of GBytes/s. The number of links used to dump the data into the system can easily reach the order of magnitude of hundreds. With respect to the computation power, it may also contain hundreds of high-end FPGA chips in order to distribute algorithm components for concurrent data processing.

- **Development difficulty.** Diverse trigger algorithms are to be implemented and optimized, corresponding to the combinational detector system adopted to investigate various particle characteristics. These algorithms often concern sophisticated principles and their implementations must satisfy certain performance requirements.

- **Design management complexity.** For such a large-scale system, the design management work is not negligible. How to reasonably partition computation tasks, to efficiently organize the interconnection network of task nodes, to map hardware implementations on different FPGA chips, and to manage design and bitstream files for hundreds of FPGAs with online and remote reconfiguration requirements, raise extra complexity to the system developers.

- **Run-time maintenance requirement.** In some circumstances, the system is expected to be online maintainable with partial functionalities kept running. A complete stop may delay the experimentation progress, and the financial cost introduced by the delay cannot be simply ignored. (With respect to the experimental facilities, even the power cost itself is an enormous number.)

- **Expensive hardware cost.** Due to the large scale of the system, hardware cost becomes a very critical issue. Optimized designs which may fully exploit the hardware resources, can actually reduce the hardware cost from another standpoint.

- **Long development cycle.** A complete DAQ and trigger system design typically lasts for 5 to 10 years or even longer.

We understand that to construct a complex DAQ and trigger system for high-energy physics applications can largely benefit from an optimal design and development methodology on FPGAs. The modern FPGA partial reconfiguration technology provides a good tool to address some dilemmas generated by the conventional static approach. Therefore we investigate the adaptive development methodology in the context of physics applications. Detailed discussion in various aspects goes in the following chapters of this thesis.

"Learning without thought is labor lost;
thought without learning is perilous."

Confucius
(Chinese educator, philosopher, and political theorist, 551 - 479 B.C.)

Chapter 3

Design Framework for Adaptive Computing

In this chapter we present a comprehensive and practical design framework for adaptive computing based on FPGA run-time reconfigurability. Several design key issues are systematically discussed in different hardware/software layers, concerning hardware architecture, run-time reconfiguration technology, OS and device drivers, hardware process scheduler, context switching as well as inter-process communications. All are the aspects which should be taken into account when practically building an adaptive system on FPGAs. The themes of this chapter cover paper [24], [27], [28], [30], and [31] listed in Section 1.4.

The modular design concept frequently adopted in static designs applies also to run-time reconfigurable designs on FPGAs. In adaptive computing, the entire system is partitioned and different tasks are to be individually implemented as functional modules. Analogous to software processes running on top of Operating Systems (OS) and competing for the CPU time, each functional module can be regarded as a *hardware process* which is to be loaded into reconfigurable slots on the FPGA rather than GPCPUs. Multiple hardware processes may share the same programmable resources and be scheduled to work according to certain disciplines on the awareness of computation requirements. Context switching happens when the current hardware process of a

task is leaving the reconfigurable slot (being overwritten) and a new task is to be loaded to start working. All these key issues in the adaptive computing framework are classified into and addressed within certain layers in hardware or software. Figure 3.1 demonstrates the layered hardware/software architecture and details in different aspects will be presented in the following sections respectively.

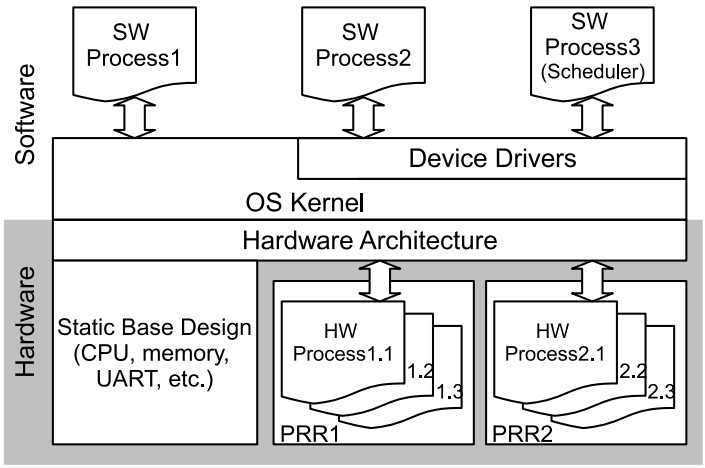


Figure 3.1. Hardware/software layers of the adaptive system

3.1 Related Work

Run-time partially reconfigurable designs become practically feasible only in the recent years, as the official technical support was announced by FPGA vendors. The PR technology is foreseen to be more and more widely used for addressing various design challenges, such as multitasking on limited resources, power reduction, cost reduction, etc. However to our best knowledge, the design framework for adaptively changeable functional modules with context-awareness is not yet well regulated in various hardware/software layers. Contributions exist concerning specific aspects of run-time reconfigurable designs.

Resource management on reconfigurable devices is a key issue of adaptive computing. In [59], a resource allocation model is presented for load-balancing

processing of multitasks. The complicated hardware architecture consisting of hierarchical Upper Management Units (UMU), Management Units (MU), Processing Units (PU) and Re-ordering Units (RU), makes it difficult and impractical for implementation. In [60], the single processor scheduling algorithm is investigated and applied to task hardware module reconfigurations. The proposed Earliest Due Date (EDD) model for synchronous tasks and the Earliest Deadline First (EDF) model for asynchronous tasks can improve the module response time when multiple designs are being alternatively loaded into multiple reconfigurable slots. Additional scheduling mechanisms and task management studies can be found in [61], [62] and [63]. However, most of the above cited investigations concentrate only on the modeling level and do not take into account practical constraints in reconfigurable designs. In [64], a practical hardware/software environment is implemented and tested to manage hardware processes on FPGAs, with the help of a modified Linux kernel called BORPH. The authors' main contribution is to enhance the OS kernel to support hardware processes and schedule them altogether with normal software processes. Nevertheless the modification work in the OS kernel space is error prone and makes the reconfigurable design dependent on the customized OS. It generates many difficulties to port the schedulable system onto different platforms.

In addition, contributions on the PR relevant issues have been reported in some other aspects: In [65], [66] and [67], run-time reconfiguration speed has been investigated and optimized, enabling practical PR designs with low reconfiguration time overhead. In [68], the authors present the approach to save and restore the hardware context of reconfigurable modules, by parsing the FPGA bitstream and extracting or recovering corresponding information. In order to interface dynamically swappable IP modules with the static system, the authors of [69] propose three wrapper designs which feature a buffer for reading out or restoring the context data of reconfigurable modules. The authors of [70] and [71] analyze the signal routing dilemmas in reconfigurable designs, and optimize communications among relocatable modules simultaneously residing in different configuration slots with a set of inter-module communication paradigms. Unfortunately the challenge of communications among hardware processes that are time-multiplexed in the same reconfigurable slot in sequence are not concerned in their work.

3.2 Overview of FPGA PR Technology

Modern FPGAs (e.g. Xilinx Virtex-4, 5, and 6 FPGAs) offer the partial reconfiguration capability to dynamically change part of the design without stopping the remaining system. This feature enables alternate utilization of on-FPGA programmable resources, therefore resulting in large benefits such as more efficient resource utilization and less static power dissipation [21]. Figure 3.2 illustrates a reconfigurable design example on Xilinx FPGAs: In the design procedure, a Partially Reconfigurable Region (PRR) A is reserved in the overall design layout mapped on the FPGA. Various functional Partially Reconfigurable Modules (PRM) are individually implemented within this region, and their respective partial bitstreams are generated and collectively initialized in a design database residing in memory devices in the system. During system run-time, various bitstreams can be dynamically loaded into the FPGA configuration memory by its controller named Internal Configuration Access Port (ICAP). With a new module bitstream overwriting the original one in the FPGA configuration memory, the PRR is loaded with the new module and the circuit functions according to its concrete design. In the dynamic reconfiguration process, the PRR has to stop working for a short time (reconfiguration overhead) until the new module is completely loaded. The static portion of the system will not be disturbed at all.

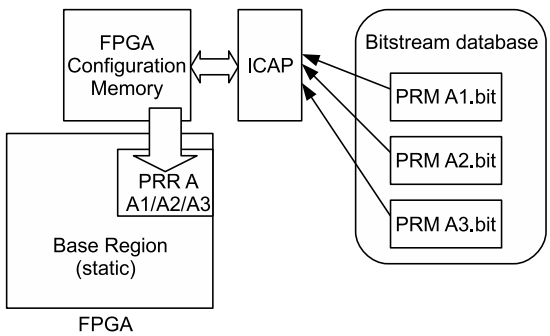


Figure 3.2. Partially reconfigurable design on Xilinx FPGAs

The PR technology is coupled very closely to the underlying framework of the FPGA chip itself. We use the Xilinx FPGAs to explain the PR design flow as illustrated in Figure 3.3: The design begins from partitioning the system between the static base design and the reconfigurable part. Usually

basic hardware infrastructures that expect continuous work and do not want to be unloaded or replaced during the operation are classfied into the static category, such as the system processor or the memory controller. The partially reconfigurable part delegates those modules with dynamically swapping needs in the PR region. All the modular designs including PRMs are assembled to form an entire system. After synthesis, netlist files are generated for all the modules as well as the top-level system. The netlists serve as input files to the FPGA implementation. Before implementation, the Area Group (AG) constraints must be defined to prevent the logic in PRMs from being merged with the one in the base design. Each PRR will be only restricted in the area defined by the RANGE constraints. Then after the following individual implementation of the base system and PR modules, the final step in the design flow is to merge them and create both a complete bitstream (with default PR modules equipped) and partial bitstreams for PR modules. Hence, run-time reconfiguration will be initiated when a partial bitstream is loaded into the FPGA configuration memory and overwrites the corresponding segment.

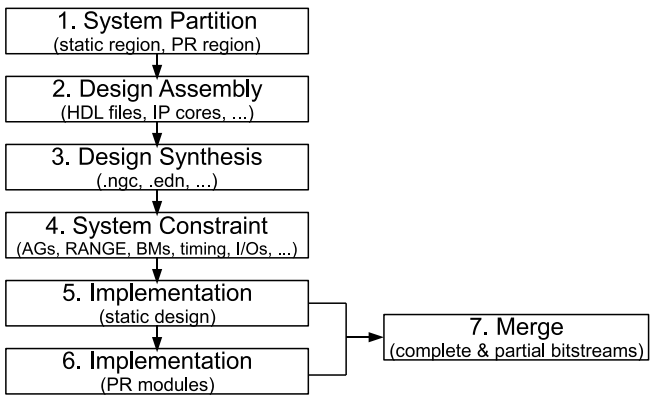


Figure 3.3. Xilinx PR design flow

3.3 Hardware Architecture

A dynamically reconfigurable design may consist of a general-purpose computer system and application-customized functional modules, as shown in Figure 3.4 for a system on a Xilinx Virtex-4 FPGA. Existing commercial IP cores can be exploited to quickly construct the general computer design. The

application-specific tasks such as algorithm accelerators feature the largest design challenges, and should be customized and optimized according to specific requirements. They are to be integrated in the system design to communicate with other components, specifically the host processor, the memory or some peripherals. In order to shorten the development period, the general computer design may be largely retained for different applications. The designers concentrate on the application-specific functionalities, customizing and optimizing the required modules.

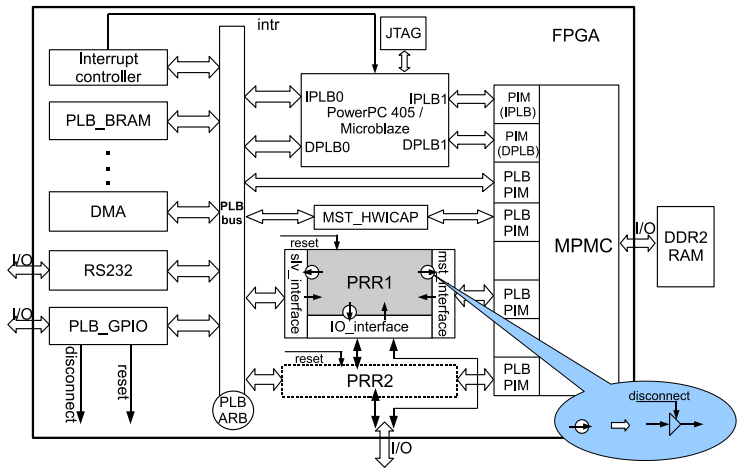


Figure 3.4. The hardware infrastructure of the PR system

In FPGA-based adaptive computing, PRRs or reconfigurable slots are reserved in the system for being dynamically equipped with different functional modules. In Figure 3.4 we show two PRRs to demonstrate the principle. When incorporated in the system design, PRMs connect to the static base design, specifically the PLB bus for receiving controls from the processor, the Multi-Port Memory Controller (MPMC) for efficiently accessing the system memory, and I/O buffers for addressing external devices. Respective interface wrappers (slave or slv, master or mst, and IO_interface) provide standard ports to connect PRMs to the base design. Direct I/O channels may also exist among various PRRs. Noting that the output signals from a PRM may unpredictably toogle during ongoing reconfiguration, "disconnect" logic (illustrated in the callout frame in Figure 3.4) is therefore required to be inserted to disable PRM outputs and isolate the unsteady signal states for the base design from

being interfered. Furthermore, a dedicated "reset" signal aims to solely reset the newly loaded module after each partial reconfiguration. Both the "disconnect" and the separate "reset" signals can be driven by a register-accessible General-Purpose I/O (GPIO) core under the control of the host processor.

In the previous Xilinx Partial Reconfiguration Early Access design flow [72], a special type of component called Bus Macro (BM) must be instantiated in the interface designs, straddling the PR region and the static design to lock the implementation routing between them. This is the particular treatment on the communication channels between the static and the dynamically reconfigurable regions. However in the new PR design flow [73], BMs are no longer needed and partition I/Os are automatically managed by the development software tool.

One significant advantage of this hardware architecture, is that it conforms to the standard Xilinx design tradition of embedded systems: The application-specific tasks are implemented into IP cores. They are wrapped by interface blocks and incorporated in the bus-based system design. Normal static designs can be easily converted into a PR system by extracting task components, combining common interfaces, and sharing the same programmable resources in a preserved PR region. Little special consideration is needed to construct a PR system on the basis of conventional static designs.

3.4 Run-time Reconfiguration Technical Support

3.4.1 ICAP Designs

The ICAP primitive is the hardwired FPGA logic by which the bitstream can be downloaded into the configuration memory. As shown in Figure 3.5, ICAP interfaces to the configuration memory and provides parallel access ports to programmable resources. During the system run-time, a master device (typically an embedded microprocessor) may transmit partial reconfiguration bitstreams from the storage device to ICAP to accomplish the reconfiguration process. The complete ICAP design, in which the ICAP primitive is instantiated, interfaces to the system interconnection fabric to communicate with the processor and memories.

As the Xilinx reference designs for PR, the structures of OPB_HWICAP [74] and XPS_HWICAP [75] are demonstrated respectively in Figure 3.6(a) and Figure 3.6(b). The OPB_HWICAP core was previously designed for the low-performance OPB bus. To link the OPB core to the system PLB, a bridge

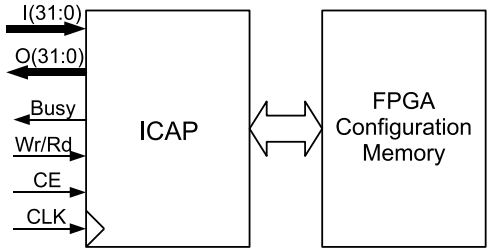
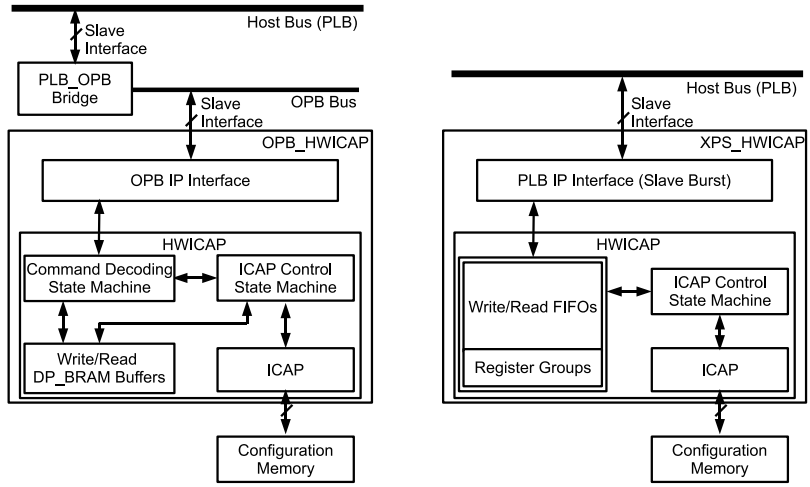


Figure 3.5. The ICAP primitive on Xilinx FPGAs

is needed for protocol adaptation. Via the OPB slave interface, the partial reconfiguration data are buffered in the Dual-Port Block RAM (DP_BRAM), if the command is decoded as "write". A control state machine diagnoses the occupancy status of the buffer and continuously supplies configuration data to ICAP, by which the FPGA configuration memory is overwritten. XPS_HWICAP shares a similar structure except that Write/Read FIFOs and register groups take the place of DP_BRAM buffers and the command decoding state machine in the OPB_HWICAP design. Also in order to increase the data transport efficiency, the PLB interface supports burst transfer mode.

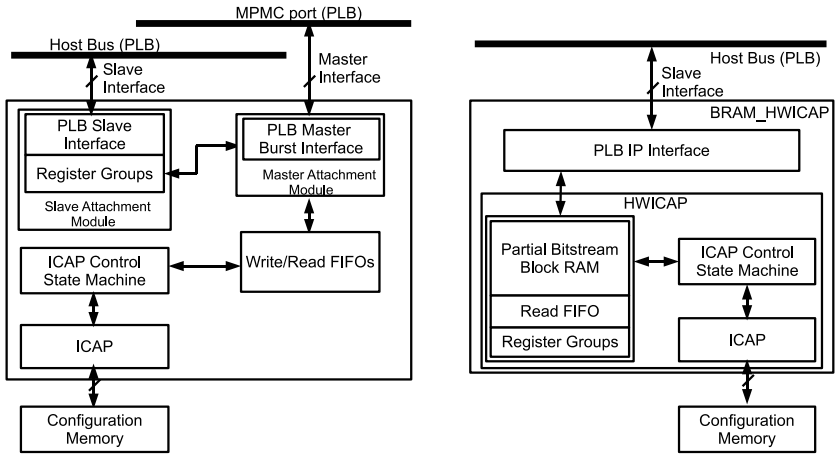
Considering the inefficiency of the microprocessor moving data to ICAP, we enhance the design with burst transmission support initiated by an integrated bus master (MST) device. The master device can actively fetch bitstreams from the system memory in a DMA mode. As shown in Figure 3.7(a), the master interface may directly couple one port of MPMC (see Figure 3.4) using the PLB protocol for efficient data movement. The slave interface is simply attached to the host PLB to receive controls from the host processor.

A particular design shown in Figure 3.7(b) is used to investigate the configuration capability of the ICAP primitive. Instead of the Write FIFO, a dedicated BRAM block is instantiated to store bitstream data. It must be large enough to hold the entire partial bitstream, because all the reconfiguration data for one PR module will be initialized there before each time reconfiguration. Partial bitstreams are alternatively loaded in BRAM through the PLB IP interface. The BRAM block works as a cache which removes the needed time to transfer data from the memory into the HWICAP module. Therefore ICAP always has its requested data ready in the high-speed BRAM. This approach can be used to evaluate the ultimate reconfiguration speed of the ICAP primitive. The design architecture is well suited for the cases in which the PR



(a) OPB_HWICAP (b) XPS_HWICAP

Figure 3.6. Structure of the Xilinx ICAP designs



(a) MST_HWICAP (b) BRAM_HWICAP

Figure 3.7. Structure of MST_ICAP and BRAM_ICAP

region is small (small bitstream files), and extremely fast reconfiguration speed is required. Its disadvantage is the high utilization of the BRAM resource on the FPGA.

3.4.2 Experimental Results

Based on the Xilinx ML405 development board [76] with a Virtex-4 FX20 FPGA, experiments have been done to measure the performance of various design structures. Processor programs are executed in the DDR memory. Partial bitstream files are initialized in DDR as well. The only exception is the BRAM_HWICAP test, in which bitstream data are initialized directly in BRAM via PLB. Different ICAP designs are employed to reconfigure a preserved PR region with the help from the host processor. In the system design, both PLB (64-bit) and OPB (32-bit) run at 100 MHz, and so do all ICAP designs. The reconfiguration time is measured from the master device starting to feed data to ICAP, and ends until the partial bitstream is completely downloaded into the configuration memory. Measured with various sizes of partial bitstream files specifically about 8, 23, 46, 80 KB, the reconfiguration speed is calculated from the recorded time span. We pick up the maximumly achieved results of all ICAP designs and list them in Figure 3.8. To demonstrate the performance dependency on the processor, we did measurements using an embedded 300 MHz PowerPC 405 microprocessor together with OPB_HWICAP and XPS_HWICAP respectively. We observe that enabling the separate 16 KB ICache and DCache of the host processor enhances the program execution speed and thus the reconfiguration speed by 17.3 and 27.3 times for two reference cores. In MST_HWICAP, the master device takes the place of the host processor to transport bitstream data. We observe that MST_HWICAP achieves a configuration speed of 235.2 MB/s, improving the data movement performance of the cache-enabled processor by one order of magnitude. We also figure out the bottleneck in MST_HWICAP to be the data delivery throughput of the DDR memory (32-bit, 100 MHz) together with the MPMC controller (100 MHz). In the design with larger DDR bandwidth or higher clock frequency, the performance result is potentially further improvable. In addition, the reconfiguration speed of MST_HWICAP is processor independent due to little processor participation in the reconfiguration work. The host processor is only responsible of initiating the storage base address as well as the data length for the integrated master device. In the last, the BRAM_HWICAP design arrives at a speed of 371.4 MB/s, which approaches the physical limit of 400 MB/s of the ICAP primitive interface (32-bit, 100 MHz). In the experiments for BRAM_HWICAP, we could not measure the

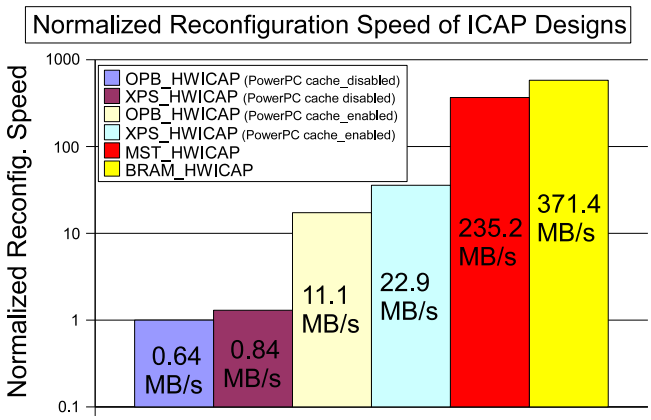


Figure 3.8. Reconfiguration performance of ICAP designs

large bitstream of 80 KB because of the constraint from the high BRAM utilization on the small Virtex-4 FX20 FPGA. A similar performance close to the physical limit of the ICAP primitive has been subsequently achieved by the authors of [67], using an external SRAM chip to replace the expensive on-chip Block RAM.

All ICAP designs have been synthesized with the Xilinx ISE and EDK v10.1 software. The resource utilization is summarized in Table 3.1. We observe that OPB_HWICAP plus the PLB-OPB bridge consumes the least resources. LUTs are mainly used as logic and one BRAM array constructs the Write/Read buffers. XPS_HWICAP utilizes much more 4-LUTs than OPB_HWICAP but zero BRAM. The reason is that its buffer devices (Write/Read FIFOs) in HWICAP are synthesized into LUTs as shift registers, rather than into BRAM. The last two columns in the table list our improved designs of MST_HWICAP and BRAM_HWICAP. We see comparable resource utilizations except for the large consumption of Block RAMs in BRAM_HWICAP. Almost half of the BRAM resource provided by Virtex-4 FX20 is used to construct the dedicated buffer for partial bitstreams. In our tests, 32 BRAMs constitute 64 KBytes address space, which implies a maximum 64 KBytes bitstream initialization in the performance experiments.

The maximum clock frequencies of all designs on -10 speed grade Virtex-4 FPGAs are listed in Table 3.2. We see our improved designs based on

Resources	OPB_HWICAP +bus bridge	XPS_HWICAP	MST_HWICAP	BRAM_HWICAP
4-LUT (total)	608 out of 17088 (3.6%)	3275 (19.2%)	1083 (6.3%)	963 (5.6%)
4-LUT used as logic	576 out of 17088 (3.4%)	907 (5.3%)	1083 (6.3%)	614 (3.6%)
4-LUT used as shift registers	32 out of 17088 (0.2%)	2368 (13.9%)	0	320 (1.9%)
Slice Flip-Flops	368 out of 17088 (2.2%)	417 (2.4%)	918 (5.4%)	469 (2.7%)
Block RAM (BRAM)	1 out of 68 (1.5%)	0	2 (2.9%)	32 (47.1%)

Table 3.1. Resource utilization of ICAP designs on Virtex-4 FX20

XPS_HWICAP do not degrade the timing performance. Both MST_HWICAP and BRAM_HWICAP feature two clock domains, one of which is the interface block and the other is the ICAP primitive. The ICAP primitive is possible to be separately clocked over 200 MHz. In practical uses, normally we choose 100 MHz for all the designs to match the clock frequency of PLB or OPB.

ICAP designs	Max. clock frequency
OPB_HWICAP/PLB-OPB bridge	248/182 MHz
XPS_HWICAP	121 MHz
MST_HWICAP	$\geq$200 (ICAP)/121 MHz
BRAM_HWICAP	$\geq$200 (ICAP)/121 MHz

Table 3.2. Timing performance of ICAP designs

We regard MST_HWICAP as the most practical IP core according to the tradeoff between the resource utilization and the performance. Therefore in the coming discussion and experiments in the remainder of the thesis, we always adopt MST_HWICAP as the FPGA run-time reconfiguration solution unless specially specified.

3.4.3 Virtual Configurations

To reduce the run-time reconfiguration time overhead has been being investigated in two directions: On one hand, design space of various ICAP modules is explored to enhance the reconfiguration throughput [31] [66] [67]. Nevertheless the reconfiguration time is still constrained by the physical bandwidth of

the reconfiguration port (ICAP) on FPGAs. On the other hand, compressing partial bitstreams has been discussed in [65], [67] and [77] for shrinking the reconfiguration time under the precondition of a fixed configuration throughput. This approach requires a compression/decompression mechanism and its corresponding hardware implementation.

As an additional solution, we propose the concept of Virtual ConFiguration (VCF) to hide the configuration overhead of a PR design. As shown in Figure 3.9, two copies of configuration contexts, each of which represents a VCF, are altogether dedicated to a single PRR on a multi-context FPGA [78] [79] [80]. The active VCF may still keep working in the foreground when module switching is expected. The run-time reconfiguration only happens invisibly in the background, loading the new partial bitstream into configuration context 2. After the reconfiguration process is finished, the newly loaded module can immediately start working by being swapped with the foreground context and migrating from the background to the foreground. The previously active configuration context will be deactivated into the background and wait for the next time reconfiguration. The configuration context swapping between the background and the foreground is logically realized by changing the control on the PRR among different VCFs. It does not really need to swap the configuration data in the FPGA configuration memory, but instead switches control outputs taking effect on the PRR using multiplexer (MUX) devices. Hence the configuration context swapping takes only very short time (normally some clock cycles), and is tiny enough to be negligible in comparison with the processing time of the system design.

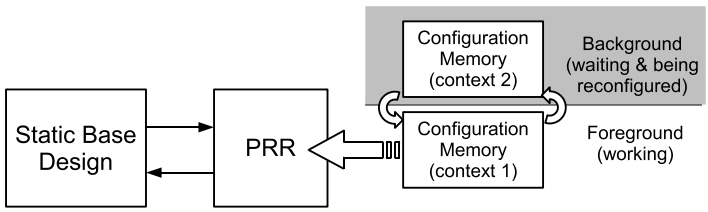


Figure 3.9. Virtual reconfigurations on multi-context FPGAs

With the approach of adopting VCFs, the reconfiguration overhead can be fully or partly removed by using duplicated configuration contexts. The timing advantage is illustrated in Figure 3.10, comparing to the canonical PR designs without VCFs. We see in Figure 3.10(a), the effective work time and the reconfiguration overhead have to be arranged in sequence on the time axis

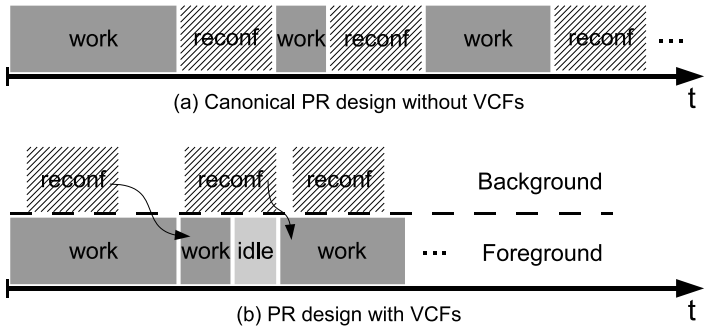


Figure 3.10. Timing diagrams of PR designs without or with VCFs

in the canonical PR design without VCFs. By contrast in Figure 3.10(b), the reconfiguration procedure only happens in the background and the time overhead is therefore hidden by the working VCF in the foreground.

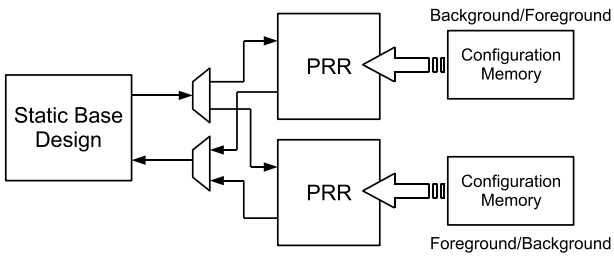


Figure 3.11. Virtual reconfigurations on single-context FPGAs

On normal FPGAs (most of the current devices) with only single-context configuration memories, VCFs may be emulated by reserving duplicated PRRs of the same size (see Figure 3.11). At each time, only one PRR is allowed to be activated in the foreground and selected to communicate with the rest static design by MUXes. The other PRR waits in the background for run-time reconfiguration and will be swapped to the foreground to work after the next scheduled module is successfully loaded. Taking into account the resource utilization overhead of reserving duplicated PRRs, usually we do not adopt more than 2 VCFs in practice.

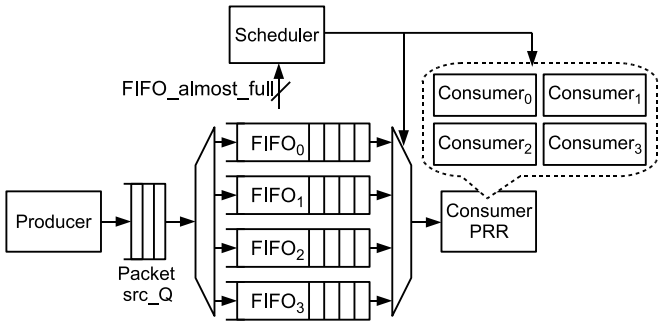


Figure 3.12. Experimental setup of the consumer-reconfigurable design

To investigate the impact of VCFs on the system performance, we set up a producer-consumer design with run-time reconfiguration capability. This design is consumer dynamically reconfigurable, meaning that multiple consumer components compete for the same reconfigurable slot, and are alternately scheduled to work in the single preserved PRR to digest their respective packets generated by the producer. As illustrated in Figure 3.12, the producer periodically injects randomly-destined packets to the four consumers and buffers them in 4 FIFO lanes. Each FIFO is dedicated to one corresponding consumer algorithm. The scheduler program[1] monitors the "almost_full" signals from all the FIFOs and arbitrates the to-be-loaded consumer module using a Round-Robin policy. Afterwards, the loaded consumer will consume its buffered data in a burst mode, until its FIFO lane is cleared and it has to be replaced by the winner of the next-round reconfiguration arbitration. The baseline canonical PR design has only one configuration context and must stop the working module before the reconfiguration starts. In the PR design with VCFs, we adopt two configuration contexts since the on-chip area overhead of multiple configuration contexts should be minimized. Experimental measurements have been carried out in cycle-accurate simulation using synthesizable VHDL codes. Simulation provides much convenience for observing all the signals in the waveform and debugging the design. It will have the same results when implementing the design on any dynamically reconfigurable FPGA. Both the baseline and the VCF designs run at a system clock of 100 MHz. The overall on-chip buffering capability is parameterized in the order

[1]To schedule hardware processes (reconfigurable modules) will be individually discussed in the coming section of 3.6 as one key issue of the design framework.

of KiloBytes. For the run-time reconfiguration time of each module, we firstly assume 10 μs which is a reasonable value when using the practical Xilinx ICAP controller for partial reconfiguration [66] [67] [31]. The generated packets are 256-bit wide. The FIFO width is 32 bits. Before packets go into the FIFO, they are fragmented into flits.

We did measurements on the received packet throughput in the unit of packets per cycle per consumer node, with the FIFO depth of 512, 1K and 2K respectively. Measurement results are demonstrated in Figure 3.13. We observe from the figure that:

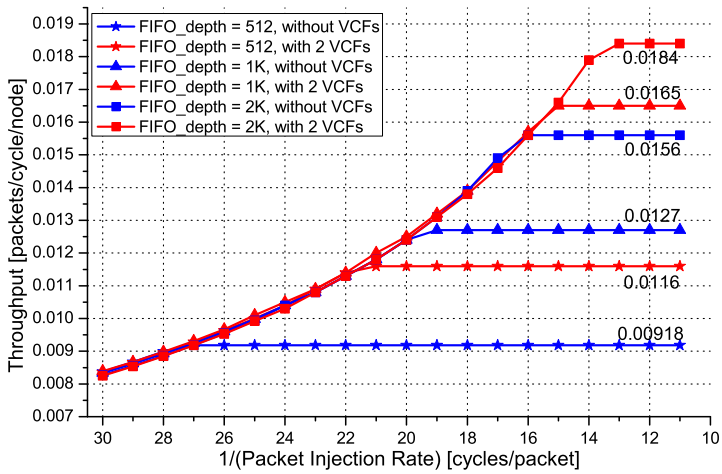


Figure 3.13. Throughput measurement results (reconfiguration time = 10 μs)

1. As the packet injection rate increases, the on-chip communication becomes progressively saturated due to the limitation of the packet consuming capability.
2. For both types of PR designs (red or light curves for with 2 VCFs and blue or dark curves for without), larger FIFO depths lead to higher saturated throughput, since the data read-out burst size can be increased by larger buffering capability, and the reconfiguration time overhead is comparatively reduced.

3. Introducing VCFs can further reduce the reconfiguration overhead by hiding the reconfiguration time in the background. In the most obvious case of 1K FIFO depth, two VCFs increase the throughput from 0.0127 packets/cycle/node to 0.0165, achieving a performance enhancement of 29.9%. Other two cases of 512 and 2K FIFO depth obtain a performance enhancement of 26.4% and 17.9% respectively.

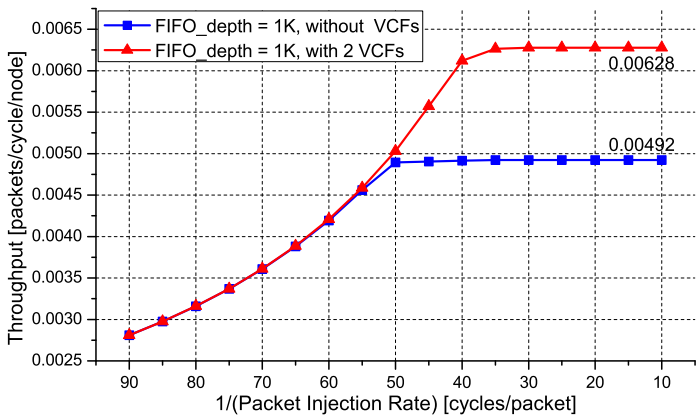


Figure 3.14. Throughput measurement results (reconfiguration time = 50 μs)

We enlarged the time span of each reconfiguration from 10 μs to 50 μs and did further throughput measurements with a middle-size FIFO depth of 1K. Results are demonstrated in Figure 3.14, comparing the PR design using 2 VCFs with the one without VCF. We observe that the overall system throughput is worsened by the increased reconfiguration time overhead, specifically from a saturated value of 0.0127 (see Figure 3.13) into 0.00492 packets/cycle/node for the non-VCF design. The increased reconfiguration time also easily results in the channel saturation at an even lower packet injection rate of about 1 packet per 50 cycles. In this test, we can still see the performance improvement of 27.6% (0.00628 vs. 0.00492 packets/cycle/node), using 2 VCFs to partly counteract the reconfiguration overhead. The channel saturation point is extended to about 1 packet per 35 cycles by the duplicated VCFs.

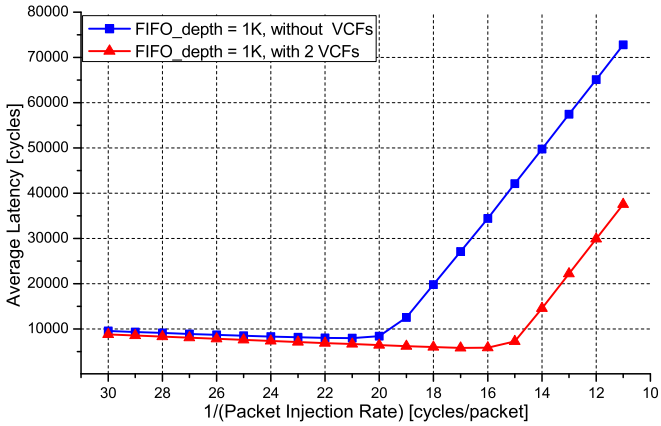


Figure 3.15. Latency measurement results (reconfiguration time = 10 μs)

In addition to the throughput comparison, we collected also statistics on packet latency performance to demonstrate the effect of using VCFs. We discuss the average latency of a certain amount of packets, and exclude the system warm-up and cool-down cycles out of the measurement, taking only into account steady communications. The latency is calculated from the instant when the packet is injected into the source queue to that when the packet is received by the destination node. It consists of two components: the queuing time in the source queue and the network delivery time in flit FIFOs. Measurements were conducted in the experimental setup with the smaller reconfiguration time of 10 μs and the middle-size FIFO depth of 1K. Results are illustrated in Figure 3.15. We observe that 2 VCFs have a slight reduction effect on the packet latency before the channel saturation. In these curve segments, packets do not stay in the source queue for too long time, but they must wait in flit FIFOs until their specific destination node is configured to read them out in a burst mode. Therefore we see two comparatively flat curve segments before the channel becomes saturated, resulting from the steady switching frequency of consumer nodes. Actually the latency segments go even slightly down along with the increment of the packet injection rate, because faster packet injection in flit FIFOs speeds up the configuration loading of consumer nodes and hence reduces the wait time of packets in the flit FIFOs. Nevertheless after the chan-

nel's packet delivery capability is saturated, packets have to spend much time waiting in the source queue to enter the flit FIFOs. Thus the average latency of packets deteriorates significantly and generates rising curve segments in the figure. By contrast, using 2 VCFs may reduce the reconfiguration overhead and extends the channel saturation to a higher packet injection rate. It reduces the packet wait time in the source queue and introduce them into the flit FIFOs at an earlier time, leading to a large improvement on the packet latency performance.

3.5 OS and Device Drivers

As in conventional static designs, all hardware modules in the reconfigurable system can be managed by the host processor with or without the OS support. In a standalone mode without OS, the processor addresses device modules with low-level register accesses in application programs. While in OSes, device drivers are expected to be customized respectively. In a Unix-like OS, common file operations are programmed to access devices, including "*open*", "*close*", "*read*", "*write*", "*ioctl*", etc. [81] Interrupt handlers should also be implemented if interrupt services are supported in the hardware design.

Different device components multiplexed in a same PR region are allowed to share the same physical address space for system bus accesses, due to their operation exclusiveness on the time axis. In fact the physical address of a device is usually fixed in the bus interface design, and therefore all PR modules have to feature the same address if they share the static interface block. In order to match software operations with the equipped hardware component, two approaches can be adopted: Either a universal driver is customized for all the reconfigurable modules sharing a same PR region. Respective device operations are regulated and collected in the code. The ID number of reconfigurable modules is kept track of and passed to the driver, branching to the correct instructions according to the currently active hardware module; or the drivers are separately compiled into software modules for different hardware components. The old driver is to be removed and the new one inserted, along with the presence of a newly loaded hardware device. Among these two approaches, the former one can avoid the driver module removing/inserting overhead in the OS, while the latter one is more convenient for system upgrade when a new task is added to share a PR region.

Little special consideration or modification effort is required on the OS and device drivers for run-time reconfigurable systems, in comparison with static designs. One important thing to remember, is to keep track of the presently

activated module in the PRR and correctly match the driver software with the hardware. Otherwise the device may suffer from misoperations.

3.6 Reconfiguration Scheduler

Analogous to the scheduler in an OS which determines the active process for CPU execution, the scheduler in FPGA reconfigurable designs monitors trigger events and decides which functional module is to be configured next in a reconfigurable slot. All hardware processes are preemptable and they must comply with the management from the scheduler. The scheduling policy may be implemented in hardware with Finit State Machines (FSM). However for more design convenience, it can be ported in software application programs running on the host processor with or without the OS support. Distinguished from the kernel space scheduling in [64] and the management unit design in hardware in [59], the user space software scheduling possesses significant advantages of convenient portability to other platforms, avoidance of error-prone OS kernel modification, and flexibility to optimize scheduling disciplines. Scheduling policies are very flexible. But they have direct effect on the system performance and should be optimized according to specific application requireiments, such as throughput or reaction latency. One general rule is to minimize the hardware context switching times, taking into account the dynamic reconfiguration time overhead and extra power dissipation needed during the reconfiguration process. Algorithm 1 and 2 demonstrate two simplified scheduler examples for throughput-aware data processing and a real-time application with fast reaction requirement: In the scheduler routine of Algorithm 1, the buffered raw data amounts of two data streams are monitored and compared. One algorithm module will be loaded into the shared reconfigurable region to digest its raw data, when there is a sufficient amount of data accumulated from the input and the amount exceeds the other stream beyond a pre-defined threshold. This scheduling policy features an intrinsic capability of balancing the processing of two data streams. It aims to minimize the overall reconfiguration overhead and guarantee the processing throughput. By contrast, the scheduler in Algorithm 2 switches immediately to the corresponding hardware module if its trigger event happens. The purpose is to react as early as possible in order to meet real-time requirements.

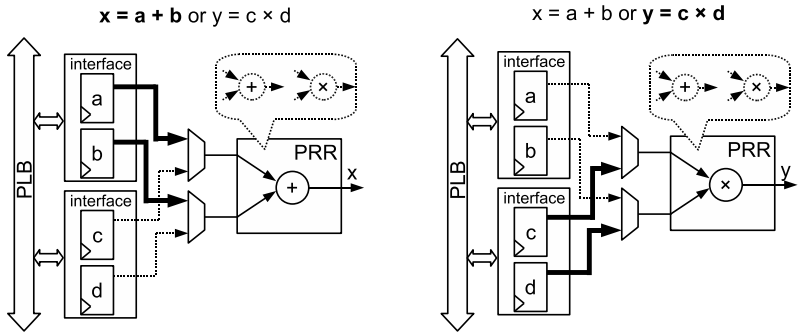
The scheduler program is only in charge of light-weight control work and usually does not feature intensive computation. In addition, the host CPU only initiates run-time reconfiguration by providing the bitstream storage address as well as the length, and it is actually the master block in the MST_HWICAP

Algorithm 1 Simplified scheduler routine example for throughput-aware processing

```
int schedule(void){
  if (data_in_buf0 - data_in_buf1) > THRESHOLD then
    switch_to_hw_process = 0; {Context switching to HW process 0, because
    there are many raw data in buffer0 to be processed.}
  else if (data_in_buf1 - data_in_buf0) > THRESHOLD then
    switch_to_hw_process = 1; {Context switching to HW process 1, because
    there are many raw data in buffer1 to be processed.}
  else
    switch_to_hw_process = switch_to_hw_process; {Keep it unchanged to re-
    duce reconfiguration overhead.}
  end if
  return switch_to_hw_process;
}
```

Algorithm 2 Simplified scheduler routine example for real-time tasks

```
int schedule(void){
  if event0_happened = 1 then
    switch_to_hw_process = 0; {Context switching to HW process 0 at once,
    since task 0 has higher priority.}
  else if event1_happened = 1 then
    switch_to_hw_process = 1; {Context switching to HW process 1, since task
    1 has also RT requirement despite lower priority.}
  else
    switch_to_hw_process = switch_to_hw_process; {No event happened and
    keep it unchanged.}
  end if
  return switch_to_hw_process;
}
```



(a) x = a + b (b) y = c × d

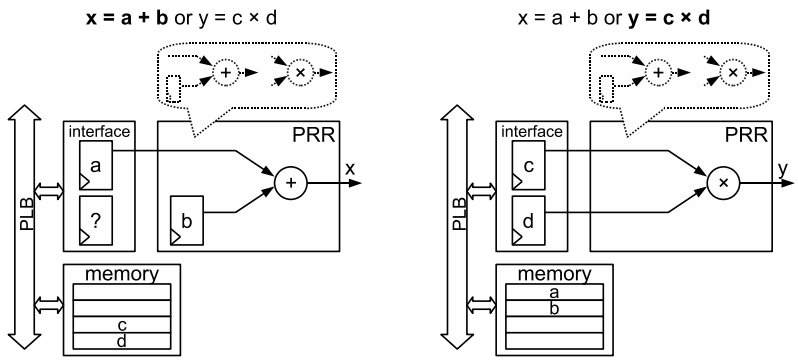
Figure 3.16. Contextless module switching in the reconfigurable design

design that transports the configuration data. Therefore dynamic reconfiguration scheduling does not take much CPU time, especially when the trigger events of module switching happen only infrequently and the scheduler is informed by CPU interrupts.

3.7 Context Switching

The context of hardware processes refers to the buffered incoming raw data, intermediate calculation results and control parameters in registers or on-chip memory blocks residing in the shared resources of PR regions or interface blocks. In some applications, it becomes contextless when the buffered raw data are completely consumed and no intermediate state is needed to be recorded. Thus the scheduler may simply swap out an active PR module, and after some time when it resumes, a module reset will be adequate to restore its operation. Otherwise, context saving and restoring must be accomplished. Figure 3.16 and 3.17 respectively demonstrate these two circumstances: In the design in Figure 3.16, two dynamically reconfigurable functional modules (adder and multiplier) do not share the bus interface and they both feature pure conbinational logic in using the PRR. Hence during each time when the PRR is configured with an arithmetic operator, the registers in interface blocks are not needed to be saved or restored in order to obtain correct results of x and y. By contrast in the design of Figure 3.17, the operand registers in the common bus interface are shared and the reconfigurable region also contains the context of one operand for the addition operation. Therefore in case of

module switching, the operands of the former operation must be saved in the system memory, and the ones for the newly resumed operator are to be restored.



(a) x = a + b (b) y = c × d

Figure 3.17. Context saving and restoring in the reconfigurable design

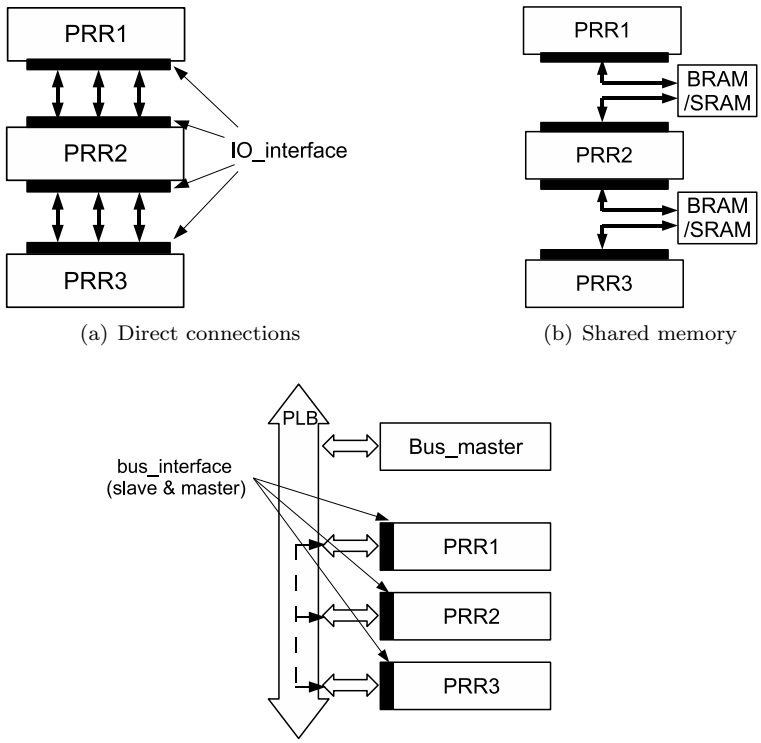
Generally speaking, two approaches can be employed to address the context saving and restoring issue: In case of small amounts of parameters or intermediate results, register accesses can efficiently read out the context into external memories and restore it when the corresponding hardware module resumes [69]. When there are large quantities of data buffered in on-chip memory blocks, the ICAP interface can be utilized to read out the bitstream and extract the storage values for context saving [68]. In order to avoid the design effort and large time overhead in the latter case, an alternative solution is to intentionally generate some periodic "pause" states without any context for the data processing module. Context switching can be then delayed by the scheduler until meeting a pause state. We will adopt this method in the application study in Chapter 6 for adaptive data stream processing.

3.8 Inter-Process Communications

3.8.1 IPC Approaches

Reconfigurable modules (hardware processes) placed at run-time typically need to exchange data among each other. In [70], the authors present four approaches to address Inter-Process Communications (IPC) among different

reconfigurable slots: direct connection, shared memory, Reconfigurable Multiple Bus (RMB), and external crossbar. We don't incline to adopt the latter two approaches in our system, considering their design complexity and large hardware overhead. But direct connections through IO_interface and shared memories can efficiently realize high-speed data delivery, as shown in Figure 3.18(a) and 3.18(b). In addition, the system bus can also be employed to exchange data in our specific system architecture. Transactions may be initiated by either a master block residing in the master interface design, or a separate master device such as the host processor or DMA. IPC via the system bus is illustrated in Figure 3.18(c).



(a) Direct connections

(b) Shared memory

(c) IPC via the system bus

Figure 3.18. IPC approaches among reconfigurable modules located in various PRRs

3.8.2 Pipe-based IPC Models

More generally, communications among reconfigurable modules that are time-multiplexed in the same PR region may also exist and be required in the hardware implementation. In previous contributions such as [70], this challenge has not been concerned. We propose a mechanism using *pipes* to deliver information among hardware processes alternately occupying the same PR region. In software, pipes and FIFOs (Also called named pipes. We use *pipe* as the general name.) are a basic IPC mechanism provided in all flavors of Unix OSes. They are best suited to implement producer-consumer interactions among processes. A pipe is a unidirectional channel: All data written by a process to the pipe is routed by the OS kernel to another process which can thus read it [82]. In FPGA-based designs, the homonymous hardware pipe behaves in an analogous manner as the software one in OSes. As an application example, Figure 3.19 shows consecutive pipe communications between algorithm modules. Conventionally all algorithm processors or algorithm steps are statically placed on the FPGA fabric. They work in parallel to process their respective input data streams, possibly generate output results, and pass IPC information to the next computation stage. In this model, intermediate pipes with buffering capability decouple and coordinate producers and consumers, if they do not generate and consume data at the same pace.

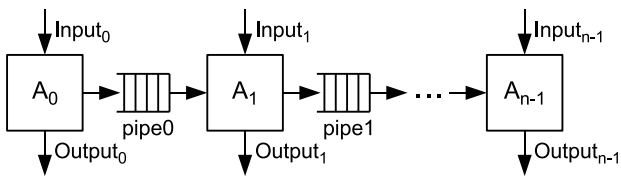


Figure 3.19. Consecutive pipe communications between algorithms or algorithm steps

In adaptive computing scenarios in which multiple algorithm processors or algorithm steps time-share the same resources in one PR region, the buffer-based pipes can be used to bridge the communication of two modules activated at different time. In Figure 3.20, a reconfigurable design is demonstrated with the same dataflow as the static algorithm placement shown in Figure 3.19. Algorithm modules are sequentially loaded into the PR region for a period of time. They read and consume data from the previous-stage pipe, and store the generated inter-module information in the current-stage pipe for next-stage

computation. For example after activated, algorithm A1 reads IPC packets of A0 from pipe0, and pass information to A2 via pipe1.

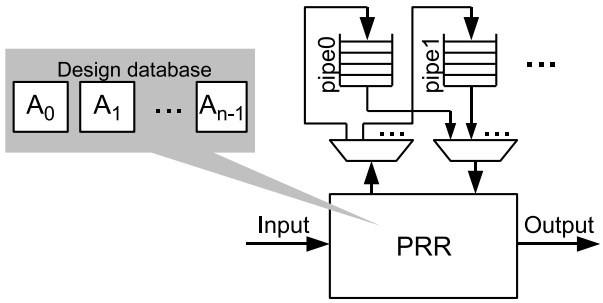


Figure 3.20. Pipe communications in PR designs

3.8.3 Performance Analysis

Referring to Figure 3.21, we introduce a metric named *Throughput Efficiency (TE)* to measure the data delivery bandwidth efficiency of the communication channel in the producer-consumer model. TE is defined as the division of the effective data delivery throughput (ε) via the channel and the minimum value of the data generation rate of the producer (ρ) and the data consumption rate of the consumer (σ), as shown in Equation 3.1:

$$TE = \frac{\varepsilon}{Min(\rho, \sigma)} \tag{3.1}$$

The minimum value of ρ and σ represents the physical limitation of data throughput in the system. The effective data delivery throughput can never exceed the smaller capability between the producer and the consumer. Therefore, the maximum value of TE is 1 (100%), which indicates that the communication channel is fully utilized and does not obstruct the data transmission flow between the producer and the consumer at all. If TE is smaller than 1, the real data delivery throughput does not reach the physical limitation. This condition implies there exists transmission efficiency loss in the channel due to some reasons.

In the static model as shown in Figure 3.19, the backpressure of intermediate buffers may coordinate the producer and the consumer to both work in the data generation or consumption rate of Min(ρ, σ). For example if ρ is

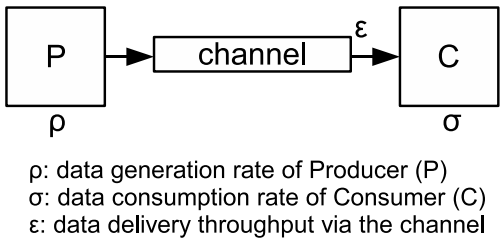


Figure 3.21. TE definition in the producer-consumer model

larger than σ (higher data generation capability than consumption), data will be backlogged in the buffer (pipe) and then either pause the producer from injecting or be overflowed. Hence the effective data delivery throughput (ε) is actually equal to σ rather than the larger ρ. Vice versa, ε is equal to ρ if the producer-consumer model features a larger data consumption capability than generation ($\rho < \sigma$). In both cases, the real data delivery throughput ε is equal to Min(ρ, σ) and TE_{stat} (static) is 1, implying no obstacle from the channel for data transmission. It may also happen that the channel cannot even handle the data rate of Min(ρ, σ), due to its physical constraints such as clock frequency limit. Thus ε will be smaller than Min(ρ, σ) and TE_{stat} will be less than 1. By contrast in the reconfigurable model shown in Figure 3.20, the pipe must be firstly filled up by the producer and afterwards emptied by consumer readout, considering that the producer and the consumer time-share the PR region and cannot work simultaneously. Assuming the size of the pipe to be S, then the data delivery throughput can be calculated with the transported data amount (equal to S) divided by the total time span for the producer to write the pipe till it is full (T_{wr}) and then for the consumer to read it (T_{rd}) till empty.

$$\varepsilon_{reconf} = \frac{S}{T_{wr} + T_{rd}} = \frac{S}{\frac{S}{\rho} + \frac{S}{\sigma}} = \frac{\rho\sigma}{\rho + \sigma} \tag{3.2}$$

Accordingly the throughput efficiency of the reconfigurable architecture is:

$$TE_{reconf} = \frac{\varepsilon_{reconf}}{Min(\rho, \sigma)} = \frac{\rho\sigma}{(\rho + \sigma) \cdot Min(\rho, \sigma)} \tag{3.3}$$

If assuming an equivalent data generation and consumption rate (ρ=σ) and no channel constraints for simplicity, TE_{reconf} is equal to 1/2 which stands for half of the value TE_{stat} in the static model. The reason comes from the fact that the producer and the consumer share the same PR region at different time, and thus the communication channel cannot be fully utilized due to the lack of concurrent writing and reading.

The above analysis is for the ideal situation. In practice, context switching overhead (T_{csp} for switching to the producer and T_{csc} for to the consumer) of dynamically reconfiguring each algorithm module should also be included in the total time span. Thus the data delivery throughput and the throughput efficiency in the reconfigurable model can be revised as:

$$\varepsilon_{reconf} = \frac{S}{T_{wr} + T_{rd} + T_{csp} + T_{csc}} = \frac{\rho\sigma}{\rho + \sigma + \frac{(T_{csp} + T_{csc}) \cdot \rho\sigma}{S}} \tag{3.4}$$

$$TE_{reconf} = \frac{\varepsilon_{reconf}}{Min(\rho, \sigma)} = \frac{\rho\sigma}{(\rho + \sigma + \frac{(T_{csp} + T_{csc}) \cdot \rho\sigma}{S}) \cdot Min(\rho, \sigma)} \tag{3.5}$$

The latency (L) of an IPC packet is the time slice from its generation by the producer till it is received by the consumer. In the static process model, the latency is simply the time needed by the consumer to fetch the packet from the pipe. Assuming that the producer and the consumer are ideally coordinated at the same pace (ρ=σ) and channel constraints are excluded, the pipe channel will behave much like a direct connection with a small delay unit and the latency can be ignored. While in the reconfigurable model, the latency consists of three compositions, as demonstrated in Figure 3.22. Firstly the packet is injected by the producer into the pipe. It stays in the pipe until the pipe is full. Afterwards the context switching reconfiguration starts and the consumer is to be activated. Finally the IPC packet is received by the consumer only after all other packets before it are read out from the pipe. Equation 3.6 lists the three latency compositions of the ith packet in the reconfigurable design, in which the pipe can hold n packets in total:

$$L_i = L_{wr\_i} + T_{csc} + L_{rd\_i} \quad (i \in [0, n-1]) \tag{3.6}$$

In the periodic traffic pattern for both the producer and the consumer, Equation 3.6 will be elaborated as:

$$\begin{aligned} L_i &= L_{wr\_i} + T_{csc} + L_{rd\_i} = T_{wr} \cdot (1 - \frac{i+1}{n}) + T_{csc} + T_{rd} \cdot \frac{i+1}{n} \\ &= \frac{S}{\rho} \cdot (1 - \frac{i+1}{n}) + T_{csc} + \frac{S}{\sigma} \cdot \frac{i+1}{n} \quad (i \in [0, n-1]) \end{aligned} \tag{3.7}$$

With a coordinated data generation and consumption rate (ρ=σ) and without channel constraints, Equation 3.6 can be further simplified as:

$$L_i = T_{wr} + T_{csc} = T_{rd} + T_{csc} = \frac{S}{\rho} + T_{csc} = \frac{S}{\sigma} + T_{csc} \tag{3.8}$$

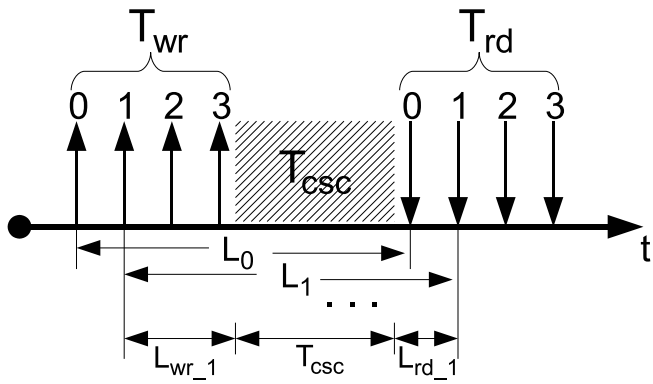


Figure 3.22. Packet latency in the reconfigurable model

3.8.4 Hardware Implementation of Pipes

The pipe can be realized by finite-depth FIFO devices using the Block RAM resource on the FPGA or external memories such as DDR SDRAM. On-chip BRAM provides more efficient data transmission bandwidth, while DDR enables larger buffering capability. Figure 3.23 demonstrates the BRAM implementation of the pipe. The producer and the consumer are dynamically reconfigured to time-share the single preserved PR region. Module switching is respectively triggered by pipe full or empty interrupts, which are detected by the scheduler program running on the host processor. The isolation logic sits between the PR region and the static design to disable the unpredictable

outputs of PR modules during reconfiguration. The separate reset signal brings a newly loaded module to its initial state. When using DDR to implement the pipe (see Figure 3.24), an interface design (ddr_pipe_interface) is required to realize DDR accesses. The interface provides the bus master functionality to actively collect produced IPC packets into the DDR memory and direct them to the consumer, via a port of MPMC. Both BRAM_pipe and the DDR memory together with its controller feature a data bus width of 64 bits at a system clock of 100 MHz.

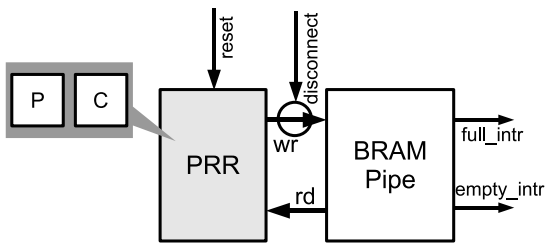


Figure 3.23. Pipe implementation with BRAM

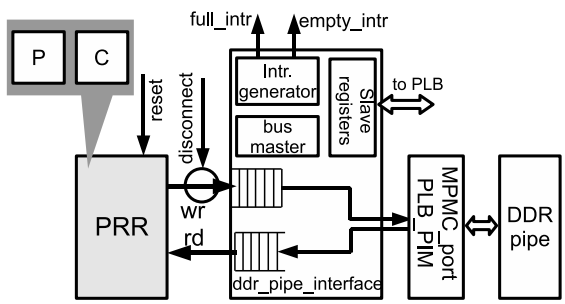


Figure 3.24. Pipe implementation with DDR

Table 3.3 lists the resource utilization of BRAM_pipe and DDR_pipe implementations. We see from the numbers that BRAM_pipe consumes the expensive BRAM resource on the FPGA to construct the FIFO. BRAM provides more efficient data delivery bandwidth, but only small pipe sizes. LUT and Flip-Flop utilizations in BRAM_pipe are negligible. DDR_pipe requires a fixed

	BRAM_pipe				DDR_pipe (with ddr_pipe_interface)					
	1 KB	4 KB	16 KB	64 KB	1 KB	4 KB	16 KB	64 KB	1 MB	16 MB
4-input LUTs	63 out of 50560 (0.12%)	59 (0.12%)	69 (0.14%)	200 (0.40%)	971 (1.92%)					
Slice Flip-Flops	53 out of 50560 (0.10%)	63 (0.12%)	73 (0.14%)	118 (0.23%)	829 (1.64%)					
Block RAM	2 out of 232 (0.86%)	2 (0.86%)	8 (3.45%)	29 (12.5%)	2 (0.86%)					
DDR SDRAM	/	/	/	/	1 KB	4 KB	16 KB	64 KB	1 MB	16 MB

Table 3.3. Resource utilization of BRAM_pipe and DDR_pipe on Virtex-4 FX60 FPGA

		Peak data gen./cons. rate (ρ, σ)	Pipe size (S, depth×width)	Partial bit-stream size (BS)	Measurement time	Delivered data amount	Data delivery throughput (ε_{reconf})	Throughput Efficiency (TE_{reconf})
Test 1	1.1	800 KB/s	128×8B=1KB	27 KB	1.427s	500 KB	350.39 KB/s	0.438
	1.2	800 KB/s	512×8B=4KB	27 KB	5.270s	2 MB	379.51 KB/s	0.474
	1.3	800 KB/s	2K×8B=16KB	27 KB	20.62s	8 MB	387.97 KB/s	0.485
	1.4	800 KB/s	8K×8B=64KB	27 KB	82.07s	32 MB	389.91 KB/s	0.487
Test 2	2.1	8 MB/s	128×8B=1KB	27 KB	0.273s	500 KB	1.83 MB/s	0.229
	2.2	8 MB/s	512×8B=4KB	27 KB	0.656s	2 MB	3.05 MB/s	0.381
	2.3	8 MB/s	2K×8B=16KB	27 KB	2.194s	8 MB	3.65 MB/s	0.456
	2.4	8 MB/s	8K×8B=64KB	27 KB	8.339s	32 MB	3.84 MB/s	0.480
Test 3	3.1	80 MB/s	128×8B=1KB	27 KB	0.159s	500 KB	3.14 MB/s	0.039
	3.2	80 MB/s	512×8B=4KB	27 KB	0.197s	2 MB	10.15 MB/s	0.127
	3.3	80 MB/s	2K×8B=16KB	27 KB	0.350s	8 MB	22.86 MB/s	0.286
	3.4	80 MB/s	8K×8B=64KB	27 KB	0.966s	32 MB	33.13 MB/s	0.414
Test 4	4.1	800 MB/s	128×8B=1KB	27 KB	0.143s	500 KB	3.50 MB/s	0.0044
	4.2	800 MB/s	512×8B=4KB	27 KB	0.151s	2 MB	13.25 MB/s	0.0166
	4.3	800 MB/s	2K×8B=16KB	27 KB	0.165s	8 MB	48.48 MB/s	0.0606
	4.4	800 MB/s	8K×8B=64KB	27 KB	0.228s	32 MB	140.35 MB/s	0.1754
Test 5	5.1	8 MB/s	128×8B=1KB	80 KB	0.500s	500 KB	1.00 MB/s	0.125
	5.2	8 MB/s	512×8B=4KB	80 KB	0.882s	2 MB	2.27 MB/s	0.284
	5.3	8 MB/s	2K×8B=16KB	80 KB	2.418s	8 MB	3.31 MB/s	0.414
	5.4	8 MB/s	8K×8B=64KB	80 KB	8.563s	32 MB	3.74 MB/s	0.468

Table 3.4. Measurement results of the pipe performance on the reconfigurable implementation

overhead of the interface design (ddr_pipe_interface). Its resource utilization is small (all below 2%) and acceptable in practical reconfigurable designs. The pipe size is flexibly adjustable in a wide range in the DDR memory.

3.8.5 Experimental Results

To verify the derived formulas, we did experiments to measure the performance and explicitly demonstrate the characteristics of pipes. For analysis simplicity, we select periodic traffic patterns and matching data generation and consumption capabilities ($\rho = \sigma$), with which TE is both analyzed and measured as 1 in the static producer-consumer design. On the reconfigurable implementation using BRAM_pipe, we measure the performance at four data rates, specifically 800 KB/s, 8 MB/s, 80 MB/s and 800 MB/s. Results are listed in Table 3.4 as the function of data generation/consumption rates (ρ, σ) and pipe sizes (S). With the same partial bitstream size of 27 KB, Test 1 - 4 correspond to slow, medium, fast, and extra-fast IPC data rates. In each test, four levels of pipe sizes are varied for measurements. We record the delivered data amount as well as the time span, and calculate the effective data delivery throughput (ε) from their quotient. According to the definition formula in Equation 3.1, we also derive the TE values of using BRAM_pipe in reconfigurable computing, and list the results in the last column of the table. To study the effect of different partial bitstream sizes (BS), which result in different context switching overhead T_{csp} and T_{csc}, we did also measurements with a larger bitstream size of 80 KB in Test 5 at a modest data rate of 8 MB/s (Practically IPCs do not feature too high rates as incoming raw data streams do). We draw TE curves in Figure 3.25 for all the tests, and observe that the measured throughput efficiency of the pipe may approach to the theoretical value of 0.5 at the end of large pipe size. In addition, a lower data rate leads to more efficient bandwidth utilization of the pipe channel. Both reasons come from the approach to reduce the context switching frequency and decrease the proportion of context switching overhead in the overall measurement time.

In FPGA-based adaptive computing, context switching overhead consists of run-time reconfiguration time for switching IP cores and software overhead, such as the time needed to stop the PRR outputs and to reset the newly loaded module. To investigate context switching overhead in depth, we compare Test 2 and Test 5 with the same data rate, and profile the composition of the overall measurement time. From the results shown in Figure 3.26(a) for Test 2.4 and 3.26(b) for Test 5.4, we observe that with a modest data rate of 8 MB/s and a large pipe size of 64 KB, effective pipe write and read operations dominate the proportion of the overall measurement time (98.24% and 95.67%

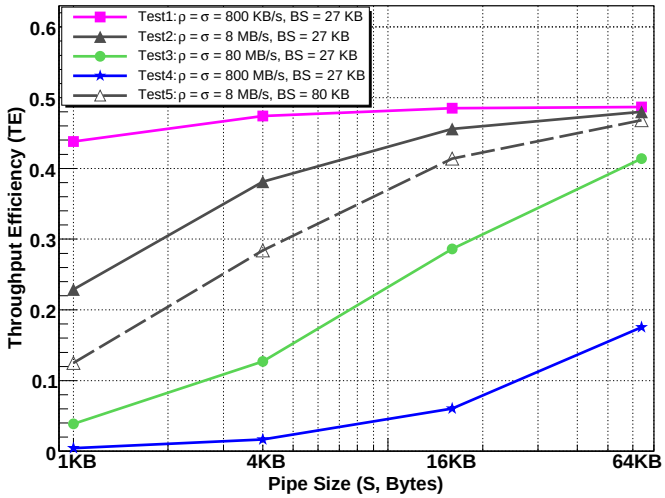
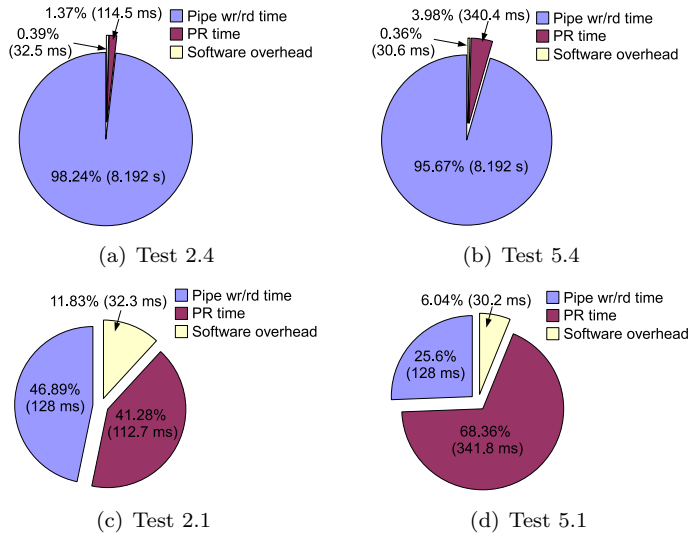


Figure 3.25. TE measurements on BRAM_pipe

respectively). Therefore the measured throughput efficiency is very close to the theoretical value of 0.5. The context switching overhead is very small in this case. Moreover from these two sub-figures, we also see the PR time as well as its proportion in Test 5.4 (340.4 ms, 3.98%) larger than in Test 2.4 (114.5 ms, 1.37%), due to the larger partial bitstream size. The software overhead is stable (∼30 ms) in both tests. By contrast, we observe much larger overhead proportions of 53.11% and 74.4% respectively in Test 2.1 and 5.1 for a small pipe size of 1 KB, as shown in Figure 3.26(c) and 3.26(d). Because of the small buffering capability, frequent context switching increases the overhead proportion and less efficiently utilizes the pipe to transport IPC data. Test 5.1 is even worse than 2.1, due to its larger partial bitstream size.

Throughput and TE are also measured on DDR_pipe. Results are illustrated in Figure 3.27. We see that DDR_pipe can improve the data delivery throughput and TE by further increasing the pipe size to 1 MB and 16 MB. With the same pipe sizes from 1 KB to 64 KB, DDR_pipe achieves similar results as BRAM_pipe, with the only exception in the test for extra-fast data rate of 800 MB/s. This is due to the situation that the memory controller bandwidth is saturated by the too high data rate and generates bottleneck to DDR_pipe accesses (i.e. physical constraint on the channel). By



(a) Test 2.4 (b) Test 5.4

(c) Test 2.1 (d) Test 5.1

Figure 3.26. Composition of the measurement time

contrast, BRAM_pipe provides sufficient bandwidth and achieves higher TE than DDR_pipe at the extra-fast IPC data rate.

Due to the time multiplexing feature of computing resources in reconfigurable designs, IPC packets cannot be directly delivered to the consumer and they have to be contained in the pipe until context switching to the consumer process. The latency evaluation has been discussed in Section 3.8.3, and we did also practical measurements on the FPGA design. In the experimental setup for BRAM_pipe and DDR_pipe, the producer transmits time tags as IPC packets into the pipe. The consumer receives them and calculates the delay from the time differences with a global timer. Large amounts of IPC packets were sampled, and the arithmetic average results are listed and illustrated in Figure 3.28 for pipes with different sizes. We observe that the latency of IPC packets deteriorates along with the increment of the pipe size (approximately linearly), as well as the decrement of the data rate. With the same pipe size, DDR_pipe results in slightly higher latency than BRAM_pipe, due to the complex DDR access mechanism: IPC packets need more clock cycles to be stored into or read out of the DDR memory. The latency of DDR_pipe apparently deviates from BRAM_pipe only at the extra-high data rate of 800

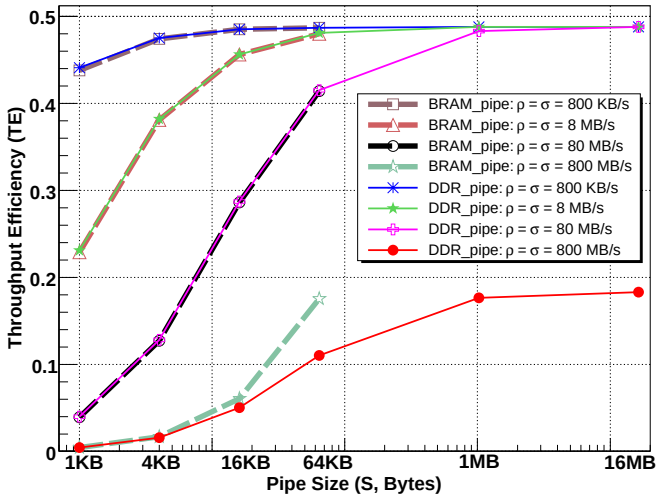


Figure 3.27. TE comparison of BRAM_pipe and DDR_pipe

MB/s, due to the saturated memory bandwidth.

There exists a pair of contradictions between throughput and latency in the reconfigurable design: To shrink the context switching overhead and increase the throughput efficiency, the pipe size is expected to be maximized. However a large pipe size leads also to worse latency performance meanwhile. For this reason the pipe size should be cautiously determined according to concrete system requirements in communication throughput or realtime performance.

3.8.6 Result Matching with Formulas

All practical measurement results on throughput, TE, and latency are verified to match very well the theoretical analysis using derived formulas in Section 3.8.3. As an instance in Test 2.1 (S=1 KB), the producer and the consumer are coordinated at a same data rate of 8 MB/s. The dedicated BRAM_pipe write and read interfaces provide sufficient bandwidth. Thus both the producer and the consumer can effectively work at the peak data rate (ρ=σ). We measured the time overhead switching to the producer (T_{csp}) and to the consumer (T_{csc}) respectively as 150.37 and 135.50 μs. Thus according to Equation 3.4 and 3.5, we theoretically derive ε_{reconf} and TE_{reconf} as 1.86 MB/s and 0.233 respectively. Both values match very well the measured

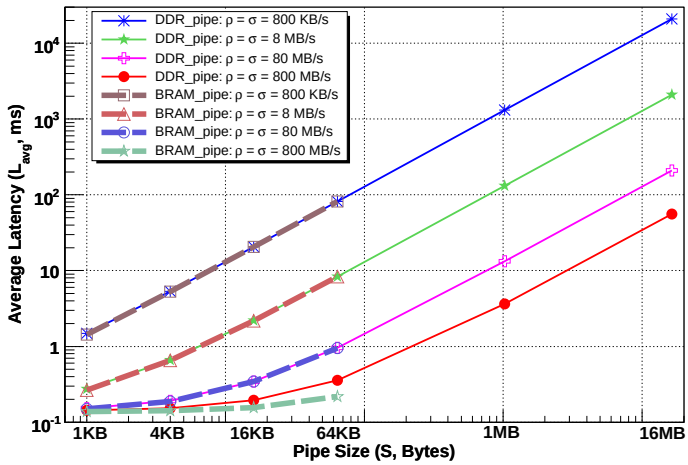


Figure 3.28. Latency measurements on BRAM_pipe and DDR_pipe

results of 1.83 MB/s and 0.229 listed in Table 3.4. With Equation 3.8 the latency is calculated as 0.261 ms, also close to the 0.266 ms in Figure 3.28. Other tests share the same analysis procedure except the ones for DDR_pipe at the extra-high data rate of 800 MB/s. In that case, the memory controller bandwidth is saturated and hence the effective data generation or consumption rate has to be lowered below the peak rate. Accordingly a pair of smaller ρ and σ results in less ε_{reconf}, TE_{reconf} and larger L_i.

"Wisdom is knowing what to do next, skill is knowing how to do it, and virtue is doing it."

David Starr Jordan
(American ichthyologist, educator and writer, 1851 - 1931 A.D.)

Chapter 4

Case Study 1: A Peripheral Controller Adaptable System

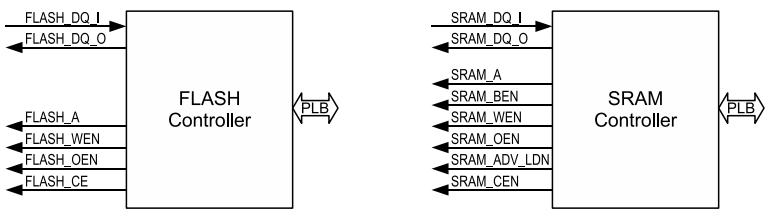
In order to verify our proposed design framework for adaptive computing, we apply the self-adaptation concept to practical embedded designs in this chapter. A NOR flash memory controller and an SRAM controller time-share the programmable resources of a reconfigurable slot on the FPGA, according to different memory access requirements. Experimental results reveal more efficient on-chip resource utilization, on the basis of accomplishing the same functionalities of flash and SRAM accesses as the conventional static design. This chapter corresponds to the published papers of [24], [26] and [30] listed in Section 1.4.

4.1 Background and Motivation

Flash memory (including NOR flash and NAND flash) is a computer storage device that can be electrically erased and reprogrammed. Due to its non-volatility, no power is needed to maintain the information stored in the chip. In embedded systems designs, flash memory is often used to store non-volatile data such as FPGA bitstreams or OS kernels. Because of its intrinsic access mode, normally it does not feature such high speed read and write operations as DDR SDRAM or SRAM.

SRAM features high-performance memory operations as well as middle-size capacity. Because of its large data bandwidth as well as short access time, it is well suited for fast and intensive random memory addressing, such as the Look-Up Table (LUT) access for our application-specific computation [41]. In many practical applications, flash memory is only used to hold non-volatile data or programs which are expected to be retrievable after each time power off. It is only addressed very occasionally or even never during the system runtime. For example, a light-weight embedded OS kernel may be loaded from flash into DDR for fast execution in case of system power-on. Afterwards, the flash memory will never be addressed in the systems operation, unless the OS kernel is scheduled to be updated. On account of the occasionality of flash accesses, it generates resource utilization inefficiency if the flash memory controller is statically mapped on the FPGA design but does not function frequently. On the other hand, our application-specific computation using SRAM as the LUT storage starts only after the FPGA firmware is configured and the OS is successfully booted. These two types of memory accesses are exclusive to some extent. Hence, we consider to make the flash memory and the SRAM time-share the same on-chip resources as well as a common device interface in the embedded design.

Both memory controller designs come directly from the Xilinx IP library. We do not have to look into their in-depth design details. Instead we regard them as blackboxes with the communication interfaces demonstrated in Figure 4.1. In the figure, the left side shows the interface to external memory devices and the right side is to the system bus.



(a) Flash controller (b) SRAM controller

Figure 4.1. Blackboxes of the flash controller and the SRAM controller

4.2 System Implementation

The reconfigurable hardware design is shown in Figure 4.2: An off-chip NOR flash memory and an SRAM share the same data, address and control bus I/O pins of the FPGA. These two chips are exclusively selected by the "CE" signal. The flash and the SRAM controllers are both slave devices on the system bus. We pick the normal controller designs out of the Xilinx IP library, and directly regard them as PR modules without any modification. The PLB bus slave interface is already included in the original design. Hence the complete IP core including the interface block is entirely fitted in the reserved PRR for run-time reconfiguration. With respect to the connections between PR modules and the base system, disconnect logic is inserted to isolate the unsteady reconfiguration state from the static design. They do not change the interface communication standards (PLB and device-specific I/O standards) and therefore little additional effort is needed to convert a conventional static design into a partially reconfigurable one.

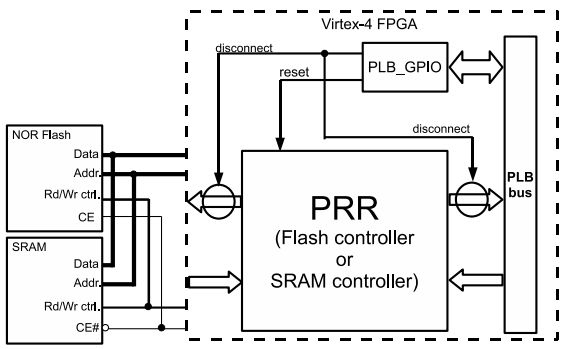


Figure 4.2. Hardware structure of the flash/SRAM PR design

An open-source Linux kernel runs on the host PowerPC 405 processor. To manage run-time operations in Linux, device drivers for hardware IP cores have been customized to provide programming interfaces to application programs. There are respective drivers needed for the flash memory, the LUT block in SRAM, PLB_GPIO and MST_HWICAP. With the drivers loaded, device nodes will show up in the "/dev" directory of the Linux file system, and can be accessed by pre-defined file operations. The drivers are compiled into modules. They will be inserted into the OS kernel when the corresponding device hardware is configured, or removed when no longer needed.

The hardware process scheduler is implemented in a C program. It detects the memory access requirements on flash or SRAM from either the system interior or external user commands, and meanwhile manages the work sequence of both types of memories. Figure 4.3 shows a flow chart, in which the scheduler alternately loads the flash and the SRAM controller with context awareness. During the device module reconfiguration, the Linux OS as well as the remaining hardware system keeps running without breaks. In this figure, steps labeled with "a - g" are used to dynamically configure the SRAM controller, and the ones labeled with "A - G" are to load the flash controller. Events marked by the symbol "≪" are detected by the scheduler to trigger hardware context switching. Main switching steps before the device operations include:

1. To save the register context of the to-be-unloaded device in DDR variables if necessary.

2. To remove the driver module of the to-be-unloaded device from the OS.

3. To disconnect the PRR outputs for isolating its unsteady state during active reconfiguration from the static design.

4. To dynamically load the partial bitstream of the expected controller by initiating the MST_HWICAP core.

5. To solely reset the newly loaded device controller, and recover its register context if there exists.

6. To re-enable the PRR outputs, restoring the communication links from the PRR to the static design.

7. To insert the corresponding device driver in the OS, for the processor access with high-level application software.

After these steps, the recently equipped controller module becomes ready for memory accesses on the NOR flash or the SRAM.

In this design, IPC operations can be realized through the third-party shared memory such as DDR. For example when the system is just powered on, the SRAM LUT initialization data are retrieved from the non-volatile flash and buffered in the system DDR memory. After the flash controller is unloaded and the SRAM controller is activated in the PRR by dynamic reconfiguration, the LUT data are then migrated into the SRAM chip for application-specific computation. The IPC data flow is illustrated in Figure 4.4.

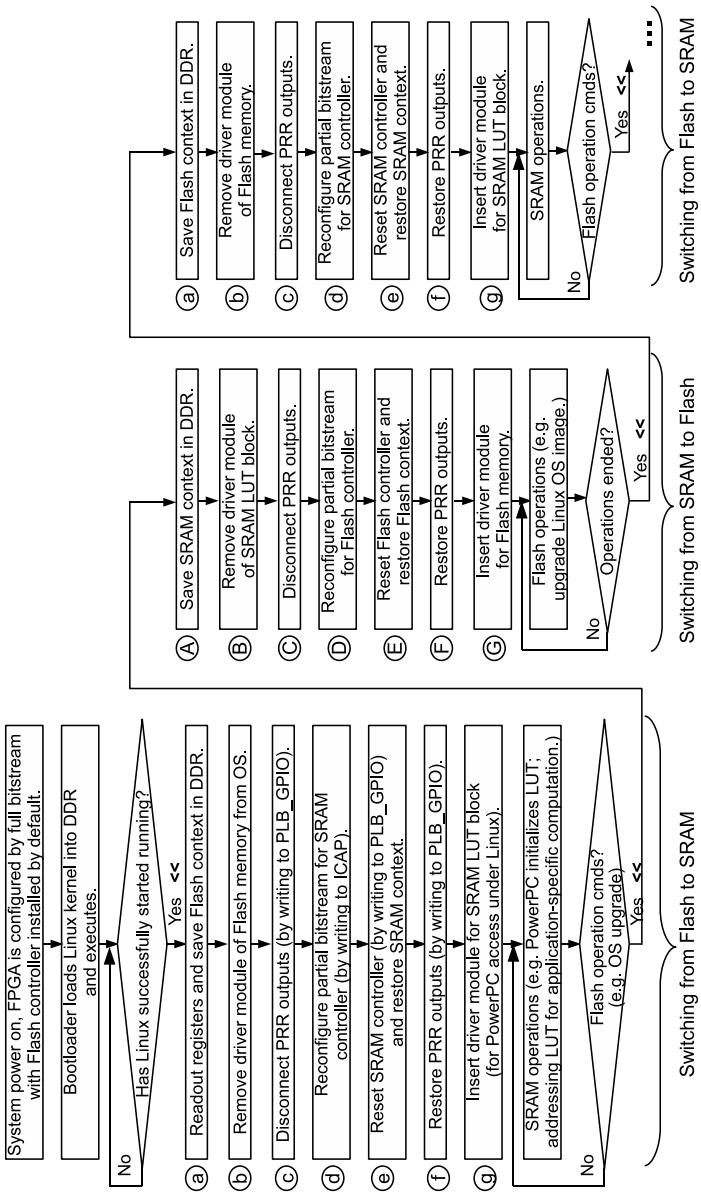


Figure 4.3. Flow chart of multiplexing flash/SRAM in Linux

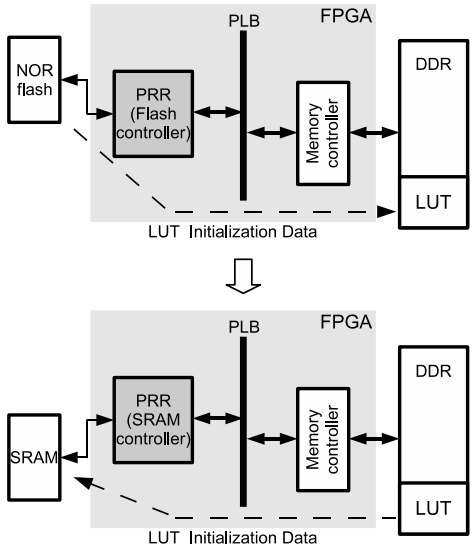


Figure 4.4. Migrating LUT initialization data from the flash memory to the SRAM

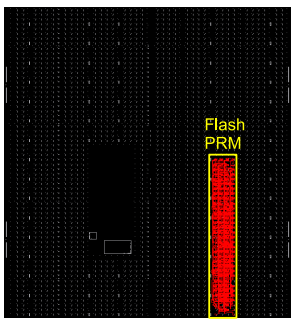
4.3 Results

Through enabling either the flash controller or the SRAM controller with system self-awareness, multitasking has been accomplished within a single reconfigurable slot on the FPGA. Figure 4.5 demonstrates the rectangular shape of the reserved PR region on a Virtex-4 FX20 FPGA layout, as well as two controller implementations after place-and-route. The reconfigurable design results in a more efficient utilization of hardware resources, as listed in Table 4.1. We understand that both the flash memory controller and the SRAM controller must be concurrently placed in the static system design, implying a total resource consumption equivalent to the sum of both device modules. A PR region is reserved in the reconfigurable design, sufficiently large to accommodate all kinds of needed resources of both device modules. Moreover, a little more resource margin is added for the place-and-route convenience of the software tool. In contrast to the conventional static approach, we observe that the reconfigurable system saves 43.7% LUTs, 33.8% slice registers and 47.9% I/O pads, with both flash and SRAM services realized. The reduced resource

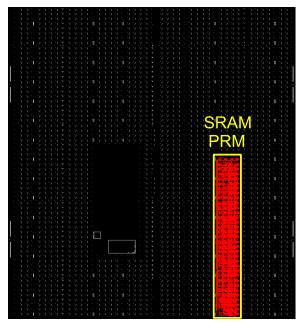
Resources	Static flash controller	Static SRAM controller	Total	PRR	Resource saving
4-input LUTs	923	954	1877	1056	43.7%
Slice Flip-Flops	867	728	1595	1056	33.8%
I/O pads	56	61	117	61	47.9%

Table 4.1. Resource utilization of the static/reconfigurable flash/SRAM designs

requirement not only enables to fit a large system design on small FPGA chips for lower hardware cost, but also makes the I/O pads shared and simplifys the PCB routing.



(a) The flash implementation in PRR



(b) The SRAM implementation in PRR

Figure 4.5. Implementation of the flash and the SRAM controller within the PRR on a Virtex-4 FX20 FPGA

"It would be possible to describe everything scientifically, but it would make no sense; it would be without meaning, as if you described a Beethoven symphony as a variation of wave pressure."

Albert Einstein
(German born American physicist, Nobel Prize winner for Physics in 1921, 1879 - 1955 A.D.)

Chapter 5

Adaptive Computing in Correlated Multi-stream Processing

This chapter focuses on the performance analysis of adaptive multi-stream processing. Correlated multi-stream processing acts as the fundamental model of our target application of data acquisition and triggering in nuclear and particle physics experiments. We set up both static and adaptive correlated multi-stream models and compare their theoretical performance by formula derivation. Experimental results demonstrate also the benefits of high performance/cost ratio as well as low resource requirements resulted from the adaptive system. The theme of this chapter relates to paper [29] listed in Section 1.4.

Stream processing is a computation paradigm in which data streams are continuously generated and flow through processing steps until final results come out. Compared to random data access architectures such as databases, stream processing results in much faster real-time response and less storage requirements for data archiving, but raises more challenges with regard to intermediate data buffering and computing capabilities to the system design [83]. For single-streaming applications, parallel computation architectures may be easily organized to meet high performance requirements. However with re-

spect to multi-stream processing with data correlation needs, complex issues on data dependency, synchronization and communication among processing units must be taken into account. Our target application of data acquisition and triggering in nuclear and particle physics experiments falls into this processing category, and hence we model the system and analyze its characteristics in this chapter.

5.1 Related Work

Computation features of stream processing have been discussed in [84], [85] and [86]. The authors analyze the microarchitecture of processors and raise their IMAGINE stream processor as an optimized solution for streaming applications, such as computer graphics or media processing. In [87], the authors propose a design optimization framework for adaptive realtime streaming applications based on reconfigurable devices. They further investigate buffer minimization and task scheduling issues for streaming applications in [88] and [89]. However their investigations are restricted to single-streams and not proper to model our specific applications. In [59], a resource allocation model is presented for load-balancing processing of multi-tasks. But the complicated hierarchical architecture makes it difficult and impractical for hardware implementation, especially with expensive FPGA resources. In this chapter, we will apply our proposed computation adaptation concept to multi-streaming applications, with on-FPGA measurement results showing the performance improvement and the ease of implementation.

5.2 Correlated Multi-streaming Models

5.2.1 Static Model

Synchronous Data Flow (SDF) [90][91] has been widely used to model and analyze streaming applications. We establish the multi-streaming model with static stream processors to appropriately describe multi-tasking applications. As shown in Figure 5.1, multiple correlated producers (P_i, $i \in [1, n]$) continuously generate data streams, and respective algorithm processors (consumer C_i, $i \in [1, n]$) digest their relevant data to obtain sub-results. Afterwards all sub-results are synchronized, correlated or assembled by the barrier synchronizer (Sync) for final results. This model is able to properly describe the DAQ and trigger system in nuclear and particle physics experiments: All data

streams originate from the nuclear or ion reactions by energetic particle collisions (corresponding to the "events" at a reaction rate of ρ in Figure 5.1). Various particle detectors (modeled as producers) generate raw data streams in response to certain interesting physics reactions. Trigger algorithms (modeled as consumers) are employed to search for expected data patterns and discard the noise on the fly to reduce the data rate. The retained interesting sub-events from all detector categories must be successfully correlated and then assembled into the pre-defined event structure for final mass storage and offline analysis (modeled as the synchronizer). In Chapter 2 we have introduced the application background in physics experiments. Figure 2.4 may be referred to for the application data flow, and it can be abstracted into the model shown in Figure 5.1.

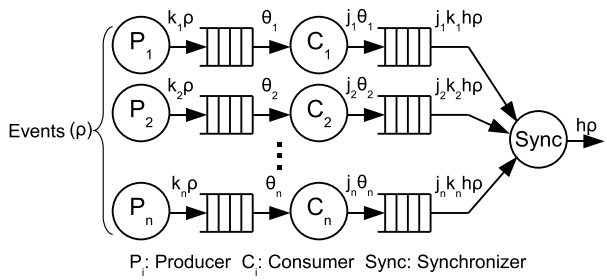


Figure 5.1. Static SDF model for multi-streaming applications

We quantify the peak data producing or consuming capabilities of different nodes by Greek letters with Latin letters as coefficients. Specifically P_i generates data at a maximum rate of $k_i\rho$. It is proportional to an overall event rate of ρ, since all producers react on the same events and strictly or statistically generate data at an approximately fixed ratio. For instance multimedia processing features statistically proportional Audio/Video data streams in the long run. Each consumer C_i consumes its respective data stream at a maximum rate of θ_i and produces sub-results at $j_i\theta_i$. Then the synchronizer collects all sub-results belonging to the same event at $j_ik_i\rho$ and recovers events which occur at the rate ρ. We may assume a higher processing capability of the synchronizer than algorithm processors. This is either true in many applications containing very complex algorithms, or can be realized by a hierarchical architecture in which the overall synchronization work is partitioned into multiple sub-synchronizers for parallel processing. This assumption aims to exclude the

synchronizer from the system bottleneck, focusing on the algorithm designs. To indicate the assumption, we scale a coefficient h ($h \rightarrow \infty$) on both input and output parameters of the synchronizer.

Along the flow paths, intermediate data are buffered in finite-depth FIFOs. When any consumer is not able to cope with the high-speed data generation, data will be backlogged in the FIFO and then either pause the producer from injecting, or be overflowed. In hardware implementations, the processing capabilities on correlated data streams are hardly guaranteed to be equivalent and synchronized, due to practical factors such as the algorithm complexity and implementation differences, system integration constraints, etc. Therefore, if excluding the synchronizer, the final result throughput is restricted by the comparatively weakest consumer which generates the performance bottleneck and leads to unbalanced processing. The system performance can be derived from the following mathematical operations: We describe the performance parameters in a matrix format with rows for streams and columns for parameters of different components as listed in Figure 5.1:

$$A = \begin{pmatrix} k_1\rho & \theta_1 & j_1\theta_1 & j_1k_1h\rho \\ k_2\rho & \theta_2 & j_2\theta_2 & j_2k_2h\rho \\ ... & & & \\ k_n\rho & \theta_n & j_n\theta_n & j_nk_nh\rho \end{pmatrix} \tag{5.1}$$

All data streams are normalized by being divided by the coefficient k_i respectively:

$$\begin{pmatrix} \frac{1}{k_1} & 0 & ... & 0 \\ 0 & \frac{1}{k_2} & ... & 0 \\ ... & & & \\ 0 & 0 & ... & \frac{1}{k_n} \end{pmatrix} * A = \begin{pmatrix} \rho & \frac{\theta_1}{k_1} & \frac{j_1\theta_1}{k_1} & j_1h\rho \\ \rho & \frac{\theta_2}{k_2} & \frac{j_2\theta_2}{k_2} & j_2h\rho \\ ... & & & \\ \rho & \frac{\theta_n}{k_n} & \frac{j_n\theta_n}{k_n} & j_nh\rho \end{pmatrix} \tag{5.2}$$

In the ideal situation when all consumers are capable of digesting their respective data streams instantly ($\{\frac{\theta_1}{k_1}, \frac{\theta_2}{k_2}, ..., \frac{\theta_n}{k_n}\} \geq \rho$), the final result throughput for the static design (P_{stat}) is equal to the event rate ρ and it presents a balanced and realtime processing capability. Elsewise ($\frac{\theta_i}{k_i} < \rho$), P_{stat} is proportionally restricted by the weakest processor on path i whose normalized data consumption rate is $\frac{\theta_i}{k_i} = \text{Min}(\frac{\theta_1}{k_1}, \frac{\theta_2}{k_2}, ..., \frac{\theta_n}{k_n})$, as formulated in Equation 5.3. In unbalanced processing of correlated multi-streams, all faster cores have to wait for the slowest stream sub-results to reach the barrier, and hence computing resources are wasted. We define the key parameter *Degree of Unbalance (DU)* as the relative stream consuming capability difference between the fastest consumer and the slowest one, as formulated in Equation 5.4.

$$P_{stat} = \frac{\theta_i}{k_i} = Min(\frac{\theta_1}{k_1}, \frac{\theta_2}{k_2}, ..., \frac{\theta_n}{k_n}) \tag{5.3}$$

$$DU = \frac{Max(\frac{\theta_1}{k_1}, \frac{\theta_2}{k_2}, ..., \frac{\theta_n}{k_n}) - Min(\frac{\theta_1}{k_1}, \frac{\theta_2}{k_2}, ..., \frac{\theta_n}{k_n})}{Min(\frac{\theta_1}{k_1}, \frac{\theta_2}{k_2}, ..., \frac{\theta_n}{k_n})} \tag{5.4}$$

5.2.2 Adaptive Model

The adaptive architecture for multi-streaming is modeled as shown in Figure 5.2. Reconfigurable slots of computing resources are reserved and they can be sequentially loaded with different algorithm processors. Flexible criteria may be adopted to select algorithms. To achieve balanced processing, algorithms can be loaded for different time spans on each slot C. The context switching overhead (mainly the reconfiguration overhead) is assumed adequately small to be neglected for analysis simplicity. We equalize the normalized data consuming capabilities of all algorithms on each slot as shown in Equation 5.5, where t_i (i∈[1, n]) stands for the effective work time of algorithm processor i within a single time unit (i.e. the time proportion of processor i occupying the slot C).

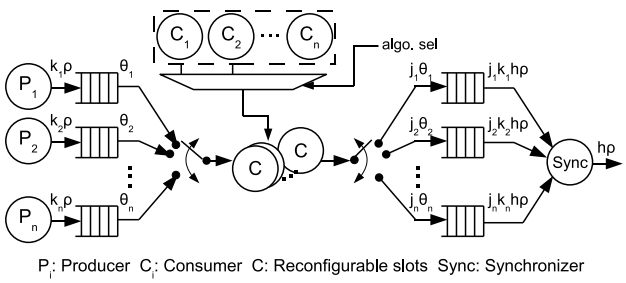


Figure 5.2. Adaptive SDF model for multi-streaming applications

$$\begin{cases} \frac{\theta_1}{k_1} \cdot t_1 = \frac{\theta_2}{k_2} \cdot t_2 = ... = \frac{\theta_n}{k_n} \cdot t_n = \psi \\ t_1 + t_2 + ... + t_n = 1 \end{cases} \tag{5.5}$$

Since the processing capabilities of all algorithms are well balanced as ψ, we may derive its value as:

$$\psi = \{\frac{\theta_i}{k_i} \cdot t_i \mid i \in [1,n]\} = \frac{1}{(\frac{k_1}{\theta_1}+\frac{k_2}{\theta_2}+...+\frac{k_n}{\theta_n})} \tag{5.6}$$

With the same number of n parallel consumer modules (same resource utilization) as in the static model, the normalized data consuming throughput for each algorithm is $\frac{n}{(\frac{k_1}{\theta_1}+\frac{k_2}{\theta_2}+...+\frac{k_n}{\theta_n})}$, also equal to the final result throughput P_{adpt}. Because we know that the arithmetic mean of a set of numbers is no greater than their maximum, as shown in Equation 5.7,

$$\frac{(\frac{k_1}{\theta_1}+\frac{k_2}{\theta_2}+...+\frac{k_n}{\theta_n})}{n} \leq Max(\frac{k_1}{\theta_1}, \frac{k_2}{\theta_2}, ..., \frac{k_n}{\theta_n}) \tag{5.7}$$

we conclude that the overall performance of balanced adaptive computing is no less than static computing (Equation 5.8). The improvement level depends on the parameter DU:

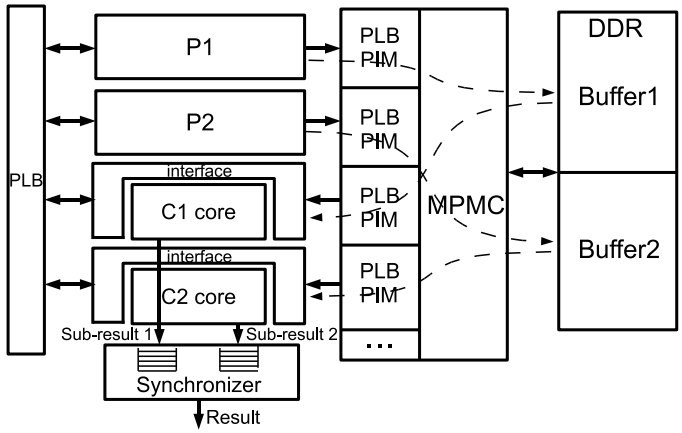
$$(P_{adpt} = \frac{n}{(\frac{k_1}{\theta_1}+\frac{k_2}{\theta_2}+...+\frac{k_n}{\theta_n})}) \geq (P_{stat} = Min(\frac{\theta_1}{k_1}, \frac{\theta_2}{k_2}, ..., \frac{\theta_n}{k_n})) \tag{5.8}$$

5.3 Experiments

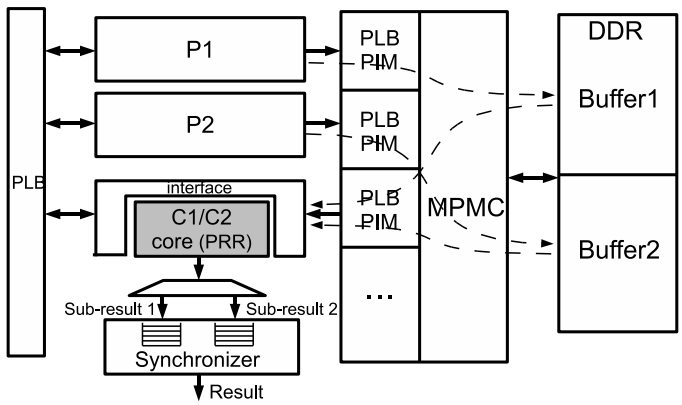
5.3.1 Experimental Setup

The multi-streaming models are implemented on a Xilinx Virtex-4 FX60 FPGA. Shown in Figure 5.3(a) for static and Figure 5.3(b) for adaptive computing, producers generate data streams and buffer them in dedicated DDR memory blocks (512 MB each). Algorithm consumers are statically or adaptively mapped on FPGA resources. The synchronizer has moderate buffering capability and correlates sub-results into final results. For analysis simplicity, we assume the following experimental setup:

1. There are two data streams with the correlation requirement.
2. Each consumer design features one unit resource utilization or on-chip area occupation.
3. The synchronizer does only simple correlation work and is excluded from the performance analysis.
4. In the adaptive design, one reconfigurable slot is reserved for implementation simplicity. Accordingly we measure and study the performance per unit area.



(a) Static computing



(b) Adaptive computing

Figure 5.3. Implementation of multi-streaming models on the FPGA

Producers and consumers are modeled with counters and they periodically produce or consume fixed-size data packets. The consumer design consists of the consumer core and a system interface, via which buffered data are fetched from DDR by an integrated bus master and the CPU releases commands to change parameters. To minimize the reconfigurable area, only the consumer core is reserved as the PR region. The interface part is incorporated in the static design.

The scheduler program for hardware management is implemented in C as a standalone application on the PowerPC processor. It keeps track of the processed data amount of each stream. Hardware processes are switched when their processed data amount difference exceeds an adjustable threshold (e.g. 8 MB in the experiments). Thus this scheduling mechanism guarantees alternate processing on multi-streams, and features an intrinsic capability to automatically balance it. The simplified scheduler is demonstrated in Algorithm 3.

Algorithm 3 Simplified scheduler routine for balanced processing

```
  {A simplified scheduler which automatically balances the processing on two
  data streams.}
char schedule(void){
  static char turn_to_which_bit = 1; {By default bitstream 1 is active to process
  stream 1.}
  if (consumed_data1 - consumed_data2) >= THRESHOLD then
    turn_to_which_bit = 2; {Context switching to hw process 2 to consume
    stream 2.}
  else if (consumed_data2 - consumed_data1) >= THRESHOLD then
    turn_to_which_bit = 1; {Context switching to hw process 1 to consume
    stream 1.}
  else
    turn_to_which_bit = turn_to_which_bit; {Elsewise the current hw process
    keeps working.}
  end if
  return turn_to_which_bit; {Return the decision to the main function.}
}
```

The consumer context includes buffered raw data in device FIFOs and address information of the DDR buffers in registers. Context switching will be postponed until all buffered raw data have been digested. In this way, computing resources are kept busy and there is no need to save the FIFO context. The register context can be read out via PLB and saved in DDR

variables. It will be restored when the corresponding consumer resumes to work.

5.3.2 Results

Referring to the normalized parameter matrix in Equation 5.2, we did normalized-setup experiments at various degrees of unbalance, as shown in Table 5.1. We choose ρ equal to 100 MB/s, which is a reasonable input data throughput for a high-speed serial communication channel such as the optical link in our practical applications. The coefficient j_i of 10^{-3} implies a data selection rate of 1/1000. Therefore the final result throughput is equal to 100 KB/s in ideal static processing when both data streams can be instantly and evenly digested by algorithm processors. In the PR design, partial bitstreams are 96 KB for both consumers. Based on Equation 5.3 and 5.8, theoretical result throughput is separately listed in the table for static and adaptive computing. Measurement results on experimental circuits involve bus conflicts and dynamic reconfiguration overhead[1], and are accordingly listed in the last table column. Considering the double-unit area utilization in the static model, both theoretical and measured result throughput per unit area are illustrated in Figure 5.4 to compare the static and the adaptive multi-streaming models. We observe that the self-schedulable adaptive computing intrinsically features the capability to balance the processing on correlated streams. It can more efficiently utilize the computing resources by keeping them less idle, and achieve a much higher performance/cost ratio (throughput-per-unit-area increased from 11.9/2 to 10.6/1 by 78.2% in Test 3) in unbalanced processing scenarios. Practical measurement results are only slightly lower than the theoretical values.

To further investigate the context switching overhead in adaptive computing, we changed the bitstream sizes and measure on the sample setup of Test 2 (DU = 3). Experimental results are shown in Table 5.2. We record the data processing time for 300 times reconfiguration. The total reconfiguration time and the software overhead (including the context saving and restoring time) are profiled and respectively listed. From the calculated percentages in the sixth column, we observe that the overall PR and software overhead takes only a tiny proportion of the overall measurement time (worst-case 0.30%). The result throughput is not exacerbated much by context switching, due to the dominant time proportion taken by the consumers to effectively process the data.

[1]Even though we have neglected the context switching overhead in the model analysis in Section 5.2.2, we do take it into account in practical experiments.

Exp. setup	Peak data producing rate of P1 (ρ)	Peak data producing rate of P2 (ρ)	Processing capability of C1 ($\frac{\theta_1}{k_1}$)	Processing capability of C2 ($\frac{\theta_2}{k_2}$)	Degree of unbalance (DU)	Data selection rate (j_1, j_2)	Results	Theoretical result throughput	Measured result throughput
Test 0 (ideal)	100 MB/s	100 MB/s	100 MB/s	100 MB/s	0	1/1000	static (area=2)	100 KB/s	95.2 KB/s
							adaptive (area=1)	50 KB/s	47.4 KB/s
Test 1	100 MB/s	100 MB/s	100 MB/s	50 MB/s	1	1/1000	static (area=2)	50 KB/s	47.6 KB/s
							adaptive (area=1)	33.3 KB/s	31.6 KB/s
Test 2	100 MB/s	100 MB/s	100 MB/s	25 MB/s	3	1/1000	static (area=2)	25 KB/s	23.8 KB/s
							adaptive (area=1)	20 KB/s	19.0 KB/s
Test 3	100 MB/s	100 MB/s	100 MB/s	12.5 MB/s	7	1/1000	static (area=2)	12.5 KB/s	11.9 KB/s
							adaptive (area=1)	11.1 KB/s	10.6 KB/s

Table 5.1. Experimental results of the static/adaptive computing performance

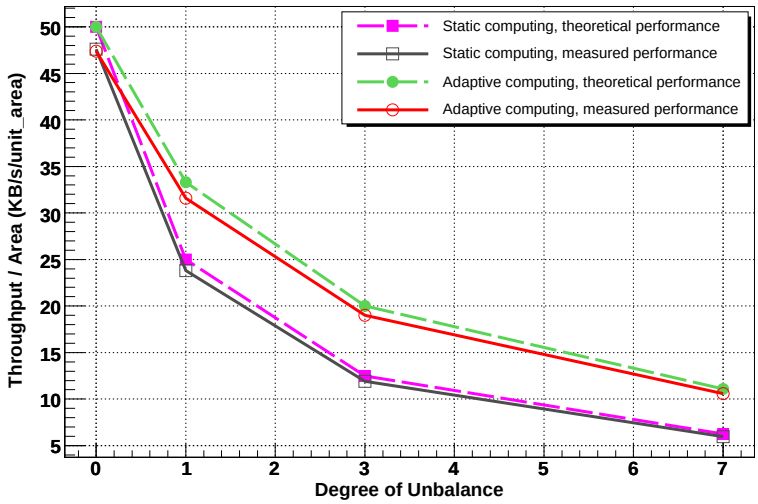


Figure 5.4. Result throughput-per-unit-area of static/adaptive computing

Partial bitstream size (KB)	PR counts	Measured processing time (s)	PR time (ms)	SW overhead (ms)	PR+SW overhead (%)	Measured throughput (KB/s)
9.6	300	125.83	12.77	187.2	0.16%	19.01
21.3	300	125.85	28.30	191.7	0.18%	19.01
61.2	300	125.90	81.24	188.8	0.21%	19.00
96.0	300	125.95	127.40	192.6	0.25%	19.00
145.3	300	126.01	192.87	187.1	0.30%	18.98

Table 5.2. Measurement results of the context switching overhead.

Although the above discussed experiments are based on two data streams, the adaptive architecture will achieve the same performance improvement effect in more complex systems with more streams. The benefits originate from the mechanism with which computing resources are efficiently kept busy on data processing instead of being idle. Performance comparison with static implementations can be estimated in practical applications with Equation 5.8 listed in Section 5.2.2.

"Religions die when they are proved to be true. Science is the record of dead religions."

Oscar Wilde
(Irish Poet, Novelist, Dramatist and Critic, 1854 - 1900 A.D.)

Chapter 6

Case Study 2: Adaptive Particle Recognition Computation

Based on the model analysis of multi-stream processing in the previous chapter, we apply the proposed adaptive design framework to our real application in nuclear and particle physics experiments. This application features two particle pattern recognition algorithms as well as their correlation requirement, fitting in the discussed multi-streaming model very well. Through adapting two algorithm engines within one reconfigurable slot on the FPGA, reduced on-chip resource requirements are observed and most processing capability of the peer static design can be retained. The content of this chapter concerns paper [24], [38], [39], and [41] listed in Section 1.4.

6.1 Application Introduction

Our target application is the pattern recognition computation for identifying particles in nuclear and particle physics experiments [49] [51]. The algorithms are implemented as hardware processing engines and integrated in the system design on FPGAs. They search for the expected patterns appearing on various particle detectors, retain only the interesting physics events generated by certain types of particles and discard the noise data on the fly.

For instance in the HADES [44] experiment, the MDC track reconstruction algorithm identifies particle flying tracks in the MDC detectors, in order to study the momenta of charged particles through their deflection in the magnetic field [41]. The Cherenkov ring recognition algorithm recognizes dilepton pairs based on the Nobel Prize winning discovery of the Cherenkov effect [92]. Detailed explanation on the physics principle as well as the hardware implementation of the MDC and RICH algorithms can be found in Appendix B. For simple clarification, we use the following Petri Net model shown in Figure 6.1 to illustrate the computation requirement.

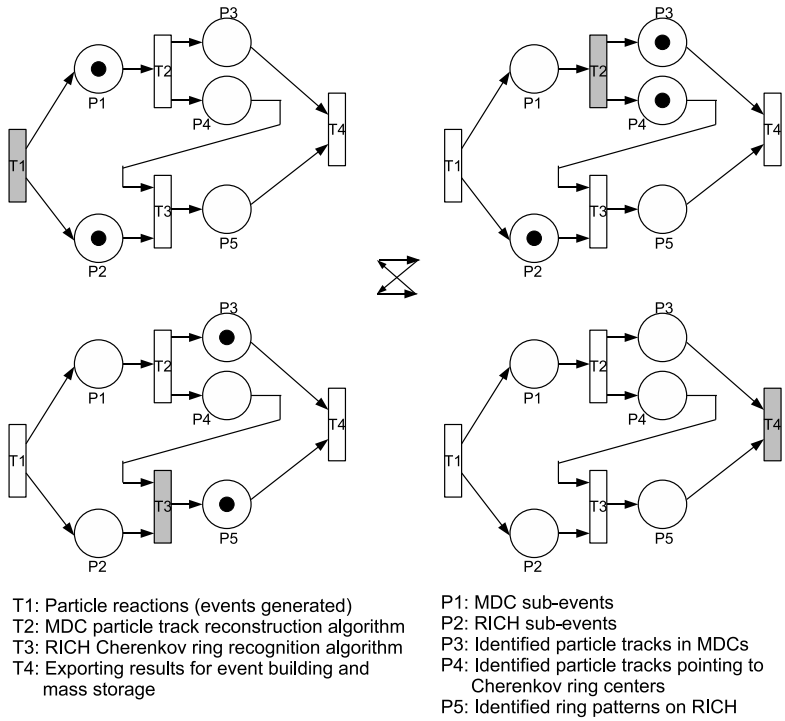


Figure 6.1. Petri Net model of the application computation

In Figure 6.1, transition T1 represents particle reactions which generate event raw data. As shown in the first step (the top-left sub-figure), MDC and RICH sub-events (position P1 and P2) are accordingly generated by MDC and RICH detectors located in the experimental facility. MDC sub-events are to

be processed by the track reconstruction algorithm (T2), searching for particle tracks penetrating the MDC detectors (step 2, the top-right sub-figure). The result of each MDC sub-event (P3 and P4) may contain identified particle flying tracks. These tracks also point to the RICH detector (P4) and specify potential ring centers around which Cherenkov rings might be recognized. In the third step (the bottom-left sub-figure), the ring recognition algorithm (T3) digests RICH sub-events and identifies ring patterns (P5), with the help from the particle tracks which imply potential ring centers. Finally (the bottom-right sub-figure) recognized particle tracks and ring patters, which belong to the same reaction event, are exported for event building and further mass storage (T4). Till now, the complete processing of a single reaction event is accomplished. In practice, incoming data are continuous and all the computing stages (transitions in the Petri Net model) work in parallel and pipeline to process the data streams.

6.2 System Implementation

Figure 6.2 demonstrates the design structure of the particle recognition system using the conventional static design approach. The track reconstruction algorithm and the Cherenkov ring recognition are respectively implemented into hardware processing engines named Tracking Processing Unit (TPU) and Ring Recognition Unit (RRU). Both cores are integrated in the system design as discussed in Figure 3.4. Incoming raw sub-events are buffered in the DDR2 memory via external optical links. They are supplied by DMA transfers into the algorithm cores via input FIFOs. With respect to the TPU module, it features a master interface via which a *projection LUT* is fetched from DDR2 for the computation purpose. Identified particle tracks which are used to point out potential ring centers are introduced to RRU through a buffer. After processing, results of both algorithms are collected back into DDR2.

In the adaptive design framework, TPU and RRU time-share the same PR region as shown in Figure 6.3. Both cores conform to the interface standard (slave and master), and therefore little modification effort is needed to port the static design into a reconfigurable one. TPU and RRU are individually synthesized and implemented using the Xilinx PR design flow. Their partial bitstreams are to be initialized in the DDR2 memory during the system runtime. We still adopt MST_HWICAP to dynamically load modules into the PR region. A hardware pipe is looped to the PR region for delivering the IPC information from TPU to RRU (i.e. identified particle tracks to point out potential ring centers). TPU uses only the uplink to write, and the downlink

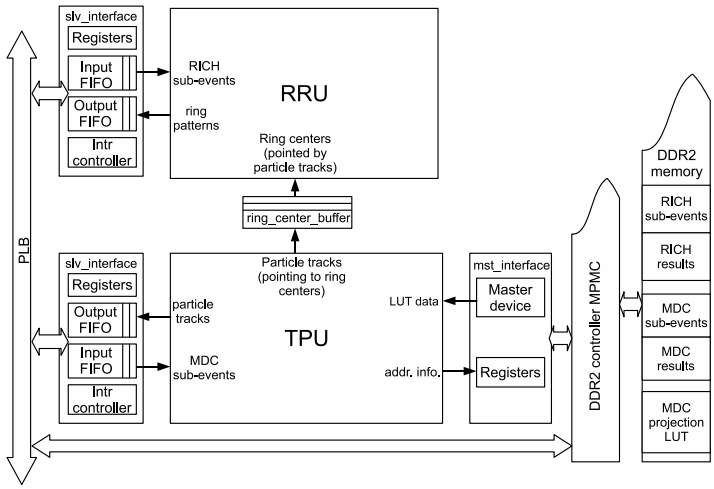


Figure 6.2. Static implementation of algorithm engines for particle recognition computation

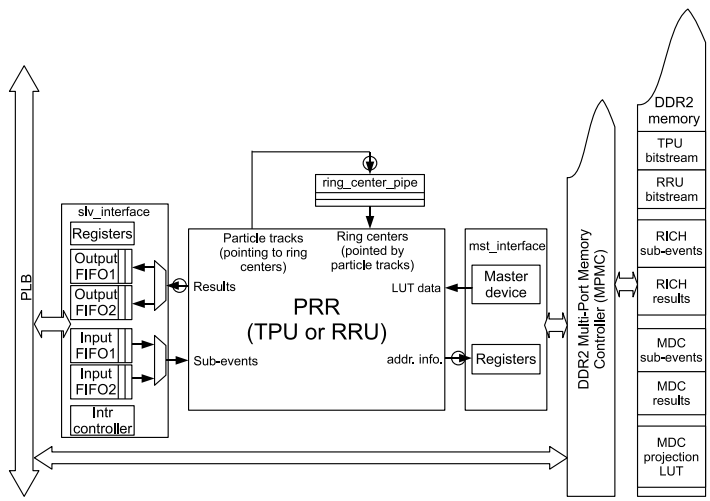


Figure 6.3. Reconfigurable implementation of algorithm engines for particle recognition computation

is only valid for RRU to read. In addition, RRU does not use the master interface.

In Chapter 5, we have analyzed the scheduling effect on the overall performance of correlated multi-stream processing with abstract models. In this practical application, we adopt the following policy to schedule TPU and RRU, in order to achieve balanced processing on particle reaction events: TPU keeps working until the *ring_center_pipe* is almost full. The *almost_full* signal informs the software scheduler running on the PowerPC processor with interrupts. The reason why we use *almost_full* rather than *full* is due to the context switching requirement, and will be explained in the coming paragraph. As the pipe is filled up, meaning that the MDC portion of certain amount of events has been processed, the scheduler will stop TPU and switch to RRU to finish RICH sub-event processing. Along with RRU's processing on RICH sub-events and consumption of the IPC data in the pipe, the *almost_empty* signal informs the scheduler with interrupts to remove RRU and bring back TPU. This scheduling policy is able to intrinsically balance the processing on TPU and RRU data streams. It keeps the resources in the PR region busy and makes their utilization efficient.

Taking into account the computation complexity and large quantities of intermediate values during data processing, TPU and RRU are believed too complicated and inefficient to save and restore their context in case of module swapping. Therefore we intentionally generate and mark some contextless states, in which PR modules can be directly swapped: Incoming raw data and processing results are respectively buffered in the input or output FIFO. In order to avoid saving/restoring this context or mixing the raw data for TPU and RRU in a same buffer, two sets of dedicated FIFOs are arranged in the common slave interface. They are statically placed and selected by the module ID currently at work. Hence the raw data and the results do not have to be swapped out in case of module switching. With respect to the computation intermediate values in the PR region, they become contextless when the current sub-event processing is completed and the next sub-event is not started yet. Context switching is thus delayed even though the scheduler has received the *almost_full* or *almost_empty* interrupt, until the current sub-event under processing is totally completed. The *almost_full* or *almost_empty* signal will also prohibit the algorithm core from further importing raw sub-events for continuous processing. For this reason, the pipe should be ready for accepting the processing results of one more sub-event after its *almost_full* signal is raised. The *almost_full* or *almost_empty* signal must inform the algorithm cores in advance, to stop their continuous sub-event processing and generate a "pause state" for contextless module switching.

	Static design					PR design			PR/static
Resources	**TPU IF.**	**RRU IF.**	**TPU**	**RRU**	**Total**	**Common IF.**	**PRR**	**Total**	**Resource saving**
4-input LUTs	992	720	5080	4008	10800	1060	5504	6564	39.2%
Slice Flip-Flops	970	730	2845	2940	7485	1093	5504	6597	11.8%
Block RAMs	2	2	46	29	79	4	46	50	36.7%

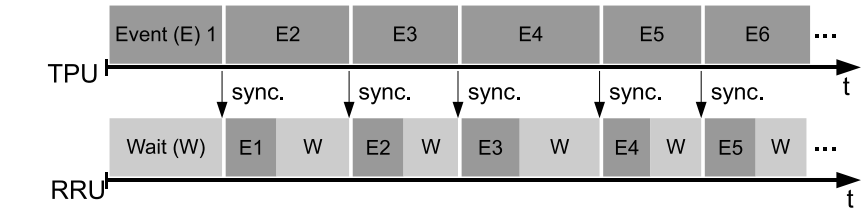
Table 6.1. Resource utilization

The scheduler software is implemented in a C program running on the host PowerPC processor. For design simplicity, we did the test in a processor standalone mode without OS support. With the device drivers of hardware modules, the scheduler program can be easily ported into an OS.

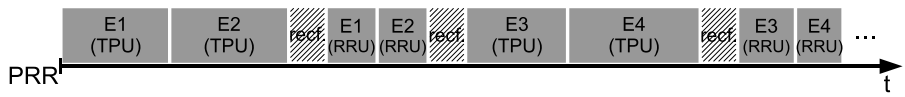
6.3 Experimental Results

Table 6.1 lists the implementation results of the static and the reconfigurable design. In the reconfigurable design, both TPU and RRU are accommodated in a PR region which is large enough to hold either module. From the last column in the table, we observe that the reconfigurable design saves 39.2% LUT, 11.8% register and 36.7% BRAM resources compared to the conventional static approach. Furthermore, the resource requirement will not increase if additional algorithm engines are to be incorporated in the system design. The scheduler can be extended to manage more algorithm modules in the same PR region. The reconfigurable design not only enables the possibility to fit the system design in a smaller FPGA chip for lower cost, but also provides the flexibility to further upgrade the system.

In terms of the performance, we initialized some event data for test in the DDR2 memory, consisting of MDC and RICH sub-events. In the static design, the TPU and the RRU module are simultaneously placed and working in a parallel fashion, as shown in Figure 6.4(a). Because of the help from TPU specifying potential ring regions on the RICH plane, the ring recognition computation load is largely reduced and RRU features a high performance. Due to the processing speed difference between TPU and RRU as well as the synchronization requirement, RRU has often to wait until it receives the IPC information from TPU. By contrast in the adaptive system, TPU and RRU are scheduled to alternately monopolize the PR region (see Figure 6.4(b)). The sequential execution mode removes the wait period and results in an efficient resource utilization. Figure 6.5 demonstrates the normalized performance



(a) Event processing in the static design



(b) Event processing in the reconfigurable design

Figure 6.4. Event processing time diagram of TPU and RRU

comparison with respect to the processing speed in the unit of Events/s. Measurements have been conducted at various pipe sizes, which directly determine the run-time reconfiguration overhead. The impact of the pipe size on reconfigurable system performance has been discussed and quantified in Chapter 3. In this case study, we observe that one time-shared reconfigurable slot in the adaptive system achieves most of the processing capability of two statically placed TPU and RRU modules, from 59.3% up to 72.8% when the pipe size ranges from 512 Bytes to 32 KB.

Pipe size	**PR counts**	**PR time**	**SW ctrl. time**	**Overall exp. time**	**Context switching overhead (PR + SW)**
512 B	2399	3.59 s	89.97 ms	18.2 s	20.22%
2 KB	599	898.5 ms	22.46 ms	16.65 s	5.53%
8 KB	119	179.3 ms	4.52 ms	12.98 s	1.42%
32 KB	23	34.5 ms	0.86 ms	10.32 s	0.34%

Table 6.2. Context switching overhead at various pipe sizes

As the increment of the pipe size, the run-time reconfiguration overhead is able to be alleviated and does not dominate in the overall processing time. Table 6.2 lists the time profiling results to demonstrate the context switching overhead: Along with the increment of the pipe size, the run-time partial reconfiguration counts happen less frequently, resulting in less reconfiguration

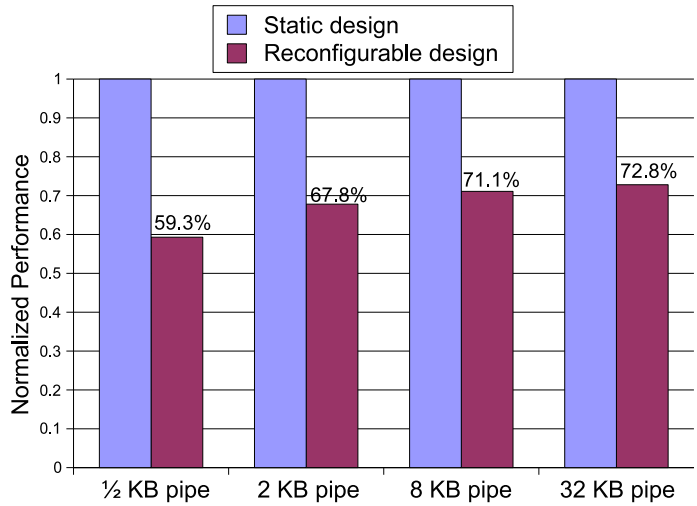


Figure 6.5. Normalized performance of the reconfigurable TPU/RRU design

time overhead out of the overall measurement time in the test. In comparison with the reconfiguration overhead, the software control overhead (e.g. to disconnect or recover the PRR outputs, to reset the newly loaded PRM, etc.) is very small. The total context switching overhead consisting of both the PR and the SW overhead takes only a negligible proportion (0.34%) of the overall experiment time at a pipe size of 32 KB. In this case, efficient event data processing dominates the system operation time, and the programmable resources in the PRR are optimally kept busy.

"Science is what you know, philosophy is what you don't know."

Bertrand Russell
(English Logician and Philosopher, 1872 - 1970 A.D.)

Chapter 7

Case Study 3: A Light-weight Routerless NoC Infrastructure

Based on the FPGA dynamic reconfigurability as well as our proposed design framework, we present a novel on-FPGA interconnection architecture in this chapter. It is regarded as an alternative interconnection method of canonical Network-on-Chips (NoC) in some scenarios especially in light-weight applications. This interconnection technology is superior to canonical wormhole NoCs in many aspects including design complexity, resource utilization, work frequency, power dissipation, etc. It is also observed to outperform wormhole NoCs in terms of packet delivery throughput in some test cases. This chapter relates to paper [25] listed in Section 1.4.

7.1 Introduction

Modern embedded applications such as mobile devices, handheld sets or multimedia terminals raise increasing challenges on computation and communication capabilities as well as power efficiency. As the great development of semiconductor technologies, immense number of transistors may be fabricated on the chip, providing the possibility to integrate very complicated systems designs. Nowadays computer systems designs have stepped into the multi-core and many-core era, integrating and interconnecting heterogeneous resources such as processors, DSPs, algorithm accelerators, memories, etc.

Both chip-level and system-level designs may enhance the computation capability by incorporating multiple processing cores, but become increasingly communication-bound. To meet on-chip communication requirements, the systems architecture has been driven to evolve from conventional bus-based towards network-based ones. Network-on-Chip (NoC) is widely considered superior to buses, in terms of performance, scalability and power consumption [93] [94] [95].

Exploiting the programmability for design convenience, NoC implementations have been extended from ASICs on modern FPGAs [96] [97] [98] [99]. The router design is usually conducted in Hardware Description Language (HDL) such as VHDL or Verilog. Hence it is straightforward to implement the NoC backbone on FPGAs with interconnected routers. IP nodes attached on routers may be either standardly obtained from commercial providers or customized according to applications. However compared to the gate-based logic implementation on ASICs, the chip area utilization efficiency and the system clock speed are very low for FPGAs, on which Look-Up Tables (LUT) are employed instead to construct combinational logics (see our early discussion on this issue in Section 1.3). Hence it is very inefficient to map NoCs on FPGAs: They have to severely underperform if such high-performance interconnection architectures are expected in FPGA-based programmable platforms.

In addition, design complexity is another unavoidable concern on implementing a complete network architecture, both on FPGAs and on ASICs [100]. The central part of the design in a network, which is the router design, covers many services to be implemented including routing, switching, channel arbitration, or even advanced Quality-of-Services (QoS). Moreover, global challenges such as inter-router synchronization, deskewed clock distribution, deadlock-free mechanisms etc., have also to be carefully considered from the network point of view. The system complexity increases the design difficulty and plays a negative role in popularizing NoCs, especially for FPGA-based applications in which most of the design is to be tailored and customized by programmers.

Based on FPGA dynamic reconfigurability, we propose a novel on-chip communication infrastructure in this chapter, which may conditionally act as an alternative of NoCs for on-FPGA systems. It features a very simple structure and does not rely on routers to deliver packets to expected destinations. Hence not only the design process is largely simplified, but also on-chip resource utilization can be significantly reduced. The operation speed and power dissipation are accordingly improved as well.

7.2 Related Work

On-chip communication infrastructures have always been a research focus and correspondingly resulted in numerous advances. In the commercial IC market, conventional arbitration-based buses are standardized for certain microprocessor architectures, for example the IBM CoreConnect bus [101] for PowerPC processors and the ARM AMBA bus [102] for ARM processors. These two buses have also been respectively ported on Xilinx and Altera FPGAs as soft IP cores, providing a systematic interconnection standard to enhance the FPGA design productivity. In [103] [104] and [105], the authors improve the bus performance by grouping the communications among bus masters and slaves, and implementing with multiple bus segments. In [106], the authors enhance the AMBA AHB with a dynamically reconfigurable topology. Up to 31.5% performance gain can be observed from their experimental results, with negligible hardware overhead. Despite these improved versions listed above, buses are still deemed inferior to NoCs, in terms of performance, power efficiency and scalability [94].

In previous contributions, dynamic reconfigurability has been considered to be adopted in the context of NoCs: In [107], the authors employ FPGA reconfigurability to adapt functional nodes (Processing Elements, PE) on the fly. In [108] and [109], the authors dynamically reconfigure the network topologies, customizing inter-node links to realize optimal interconnections for applications with different logical topology requirements. In [110] and [111], the design complexity on reconfigurable switching, routing, and data encoding strategies in network and transport layers is compensated with improved system performance and transmission reliability. In the router design in [112], the authors set up VIP connections over one virtual channel and bypass the entire router pipeline for some packet traffics with high bandwidth requirements. The VIP circuits can be either constructed based on the application task graph at design time, or dynamically configured during system run-time by monitoring the NoC traffic. The above listed contributions are all based on interconnected routers to deliver packets hop by hop (partly with shortcut links in [112]) and cannot avoid the design complexity.

Distinguished from the conventional data delivery models of buses, crossbars, and interconnection networks, etc., the authors of [113] propose a novel concept called “Move Logic Not Data (MLND)”. In an MLND system, data do not have to be delivered among various processing steps. Instead, data wait in partitioned memory pools while various logic modules (algorithm processors) are brought to the data to conduct transformation functions in turn. Typically logic modules may share the on-chip area through reconfiguration. Based on

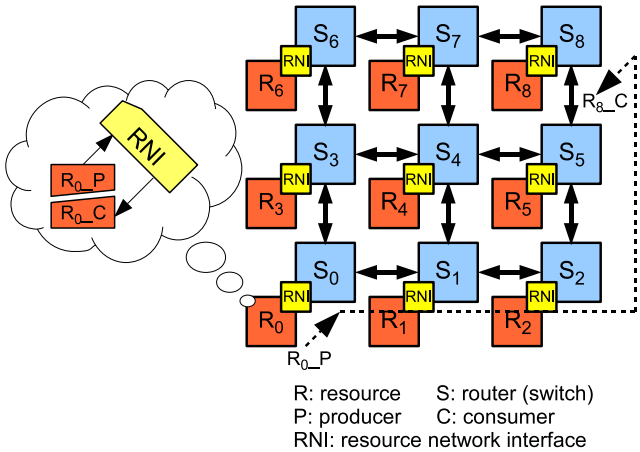


Figure 7.1. A typical 2D-mesh network architecture

the MLND concept, we thoroughly break away from the canonical router-based architecture in NoCs and present a virtually point-to-point on-chip interconnection infrastructure: Data movement is minimized in the system. Source and destination nodes are respectively multiplexed to create/digest data using FPGA dynamic reconfigurability. Various source-destination pairs time-share a same physical channel for packet delivery. We will come to more details of the architectural discussion and comparison in Section 7.4.

7.3 Canonical NoC Architecture

Networks are usually classified into two categories: circuit-switching and packet-switching. Compared to the circuit-switching network in which dedicated links are set up and reserved for transactions, packet switching with buffering capability can more efficiently utilize links and provide higher communication throughput. On-chip networks target on high performance applications and are mostly inclined to adopt packet-switching. Figure 7.1 demonstrates a widely adopted 2D-mesh topology in many packet-switching NoCs. The network architecture is constructed by interconnected routers (or switches, S). Different types of resources (R) interface to the network through Resource Network Interfaces (RNI), and eject packets to or receive from other nodes via

hops of routers. For example, when R0 intends to talk to R8, packets may go along the path of R0→S0→S1→S2→S5→S8→R8.

Figure 7.2 illustrates a canonical input-queuing wormhole router architecture [43] [114]. A similar model has been proposed in [115], and used for analyzing the delay feature of routers in [116]. The Æthereal NoC [117] also features a similar design structure in its Best-Effort (BE) architecture. The router has p Physical Channels (PC) and v Virtual Channels (VC) per PC. It conducts credit-based link-level flow control to coordinate packet delivery between adjacent routers to avoid buffer overflow and flit loss.

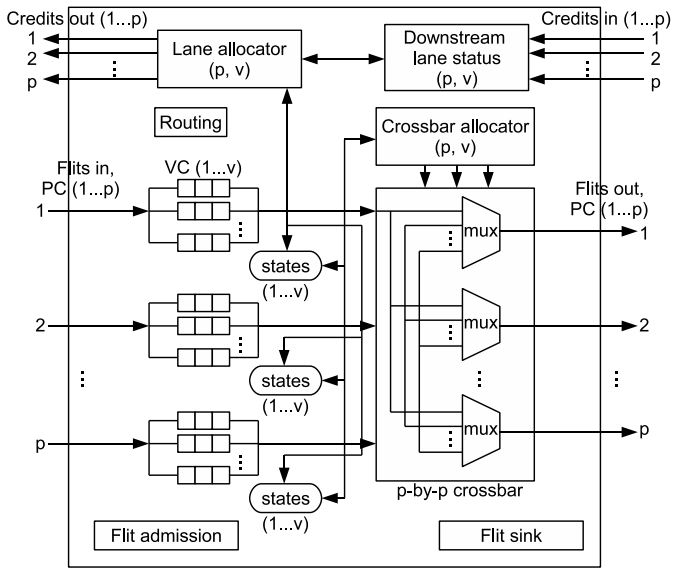


Figure 7.2. A canonical wormhole router structure

7.4 Light-weight Routerless NoC

7.4.1 Fundamental Principle

We propose a time-multiplexingly point-to-point on-chip communication infrastructure, taking advantage of the FPGA dynamic reconfigurability and the MLND concept. We name it RouterLess NoC (RL-NoC) to distinguish

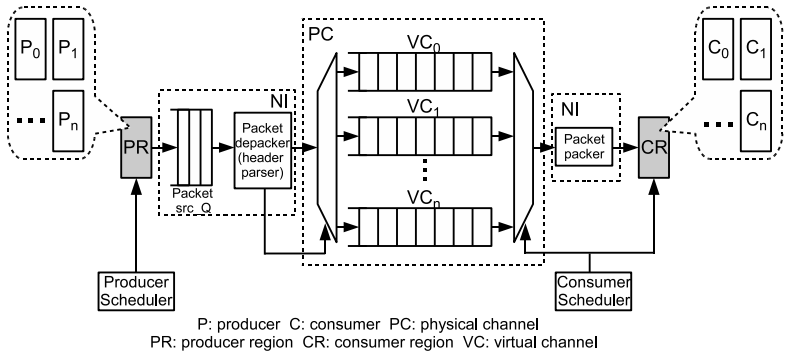


Figure 7.3. The light-weight routerless NoC

it from the canonical router-based NoC. It does not contain routers in the network, but is able to realize all kinds of logical interconnections among multiple nodes provided by canonical NoCs. RL-NoC reserves a simplex Physical Channel (PC) and dynamically bridges various source and destination nodes in an alternate way. As shown in Figure 7.3, one Producer Region (PR) and one Consumer Region (CR) are reserved as partially reconfigurable regions on the FPGA. All the resource nodes can be functionally split into producer and consumer (see Figure 7.1), which are to be respectively loaded in PR or CR during the system run-time. The PC consists of n Virtual Channels (VC) corresponding to n consumer nodes, being simply implemented with buffer devices (FIFOs). Each VC is designated to one consumer node for collecting its destined packets from all the producers. A producer scheduler and a consumer scheduler monitor the system status and respectively manage the switching of producer or consumer nodes. The consumer scheduler also accordingly selects the VC to be read out, since each consumer node at work in CR possesses its own dedicated lane for flit buffering. The Network Interface (NI) consists of a packet depacker module in the producer end and a packet packer in the consumer end. The packet depacker segments the queued injected packets into flits, and selects the VC lane to be written into by parsing the destination address section (dest_addr) in the packet header (refer to Figure 7.8 in Section 7.6.1 for the packet structure used in the experiments). The packet packer restores the packet format from flits for the consumers. In this RL-NoC model, a back-pressure flow control employs the "full" signals from VC buffers to prevent the producer from further injecting packets into a VC. Moreover,

the "almost_full" signal of the VC will invoke the loading of its corresponding consumer to digest the packets in this buffer (see the coming scheduling policy discussion for details). With respect to the routing scheme, it is simply realized by sorting packets to designated lanes of various destinations. Deadlock and livelock possibilities are completely excluded by the single-hop routing scheme. The current RL-NoC model conducts best effort deliveries, and QoS mechanisms can be further extended to provide differentiated or guaranteed services.

The source and the destination nodes are decoupled by asynchronous flit FIFOs in RL-NoC. They may have respective clock domains for FIFO write and read. In addition, various reconfigurable modules which share the same reconfigurable region (PR or CR) are also allowed to feature different clock domains [73]. This globally asynchronous clocking scheme provides much simplicity for clock distribution consideration in the design.

In order to investigate the basic characteristics of RL-NoC, we separate the producer and the consumer for simplifying the model and excluding node design details from the network analysis. In practice, for example in many streaming applications, the consumer and the producer are often tightly coupled in a unified node design: The node digests incoming data from its preceding stage, and meanwhile generates results which are to be used by the next stage task. In this case, the reconfiguration unit becomes the unified complete node design (see Figure 7.4) which obtains both downlink and uplink from and to the network. The unified node can be technically achieved by simultaneously loading the consumer-producer pair in CR and PR, with their internal communications realized through the links crossing these two reconfigurable regions [73]. Hence the node is able to read out its incoming data from one VC of the network and return results back to another. In the rest of the chapter, we adopt the simple but general producer-consumer model to focus our study on the RL-NoC infrastructure.

7.4.2 Scheduling Policy

In the specific RL-NoC model, the PR and the CR obtain their respective schedulers. In our experimental studies, all the producers are fairly scheduled to inject packets for equivalent time slots, so as to generate a general stimulus for the RL-NoC investigation. The scheduling policy of producers is to be tailored in practical applications, according to the concrete traffic generation pattern or data incoming pattern. Along with packets injected into the VCs, a certain consumer node will be marked as "to-be-loaded" when its dedicated VC buffer is almost full. The software scheduler of consumers will be then

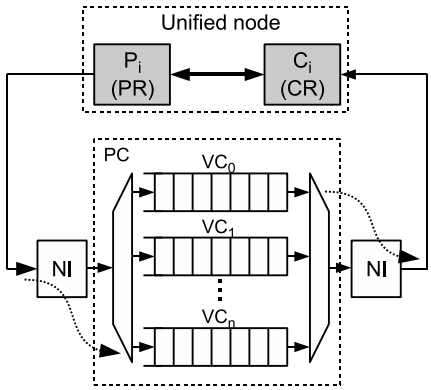


Figure 7.4. Node design with coupled producer-consumer in RL-NoC

informed with the data availability in that VC by CPU interrupts. The to-be-loaded consumer module waits until the currently working consumer finishes reading out its buffered flits, and then is configured to function by loading the partial bitstream. In case of multiple to-be-loaded consumers competing for utilizing the hardware resources in the reconfigurable CR, they are designated as the winner in a Round-Robin fashion. This consumer scheduling policy can largely utilize the buffering capability of VCs and reduce the module switching overhead. Thus the packet delivery throughput is able to be maximized, resulting in a system design proper for large-scale streaming processing in our target applications. On the other hand, packet latency performance is deteriorated since large chunks of data have to wait in the VC buffers for being read out in a burst mode. Experimental results using this throughput-aware scheduling policy will be discussed in Section 7.6.1.

7.4.3 Comparison with Other Communication Architectures

In the RL-NoC model, multiple producers and consumers time-share the PC to inject or retrieve packets for data exchanging. All kinds of logical interconnections can be realized by dynamically allocating the PC front-end to producers and the back-end to consumers. Dedicated flit buffer lanes work as mailboxes to guide packets to their expected destinations. The concept of Time-Division Multiplexing (TDM) has been widly employed to act as

the fundamental principle in various communication approaches, for instance canonical buses [101] [102], crossbar switches [119] [120], and circuit-switching networks [121] [122]. Although sharing the principle of time-multiplexing multiple nodes in using the communication channels, RL-NoC is believed to essentially differ from the above mentioned architectures. Firstly, RL-NoC removes the address bus, and avoids the lane allocation and arbitration overhead for channel occupation in networks. Secondly, it does not require all the nodes to be statically placed and interconnected, therefore leading to more efficient resource utilization as well as power consumption on FPGAs. This feature is especially gainful in the situations, in which some resource nodes operate exclusively or only occasionally. In addition, flit FIFOs decouple data transmitting and receiving, thus differing RL-NoC from bufferless communications (buses, crossbars and circuit-switching networks) in which PCs are completely reserved for a source-destination pair. As packet-switching does, the buffered communication not only improves the channel utilization efficiency, but also saves the link setup time which is respectively the arbitration time for buses and crossbars or the path setup time for circuit-switching networks. Therefore due to the existence of packets and flits in the data delivery process as well as a similar principle of decoupling transmitting and receiving with buffers, it is more meaningful to compare RL-NoC with packet-switching NoCs.

7.4.4 Performance Scaling

To meet the bandwidth requirement from different applications, one PC may not be sufficient to provide the expected data delivery throughput due to the time-multiplexing feature of multiple nodes. In this case, the infrastructure shown in Figure 7.3 can be simply duplicated to provide parallel channels for increased bandwidth (shown in Figure 7.5). When duplicated PCs are completely independent with each other, the overall bandwidth will be linearly increased along with the count of PCs. In fact, the performance can be even more than linearly scaled by optimally grouping the communications among the nodes. For example Figure 7.6(a) shows the logical topology of an application example which is to be implemented in circuits. In the figure, Rs represent resource nodes or tasks, and they are assumed to be dividable into the producer and the consumer part. We use bold arrows to indicate high-bandwidth communication requirements. When mapping the topology on a homogeneous 3×2 2D-mesh NoC, one optimal solution is to place the heavy-traffic node R_5 and R_6 in the middle, as shown in Figure 7.6(b). However traffic congestion will still probably happen on the links close to R_5 and R_6, unless they are intentionally enhanced with high bandwidth. By contrast in

a channel-doubled RL-NoC solution shown in Figure 7.6(c), intelligent communication grouping may be conducted as the following: To guarantee the bandwidth requirement for R_5 and R_6, they monopolize one PC and the rest nodes with low bandwidth requirement share the other one. Corresponding to the number of reconfigurable consumer nodes, only 2 or 4 VCs are respectively needed to collect destined packets in either PC. In addition, only the producer nodes which have communication relationship with the consumer nodes are to be included in the producer design database of PR. Fewer producers/consumers using the monopolized PC reduce the dynamic reconfiguration overhead needed to switch tasks, and result in an effective throughput scaling more than linearly. Moreover the resource utilization with the grouping approach is less than linearly increased because of the reduced count of flit lanes in each PC.

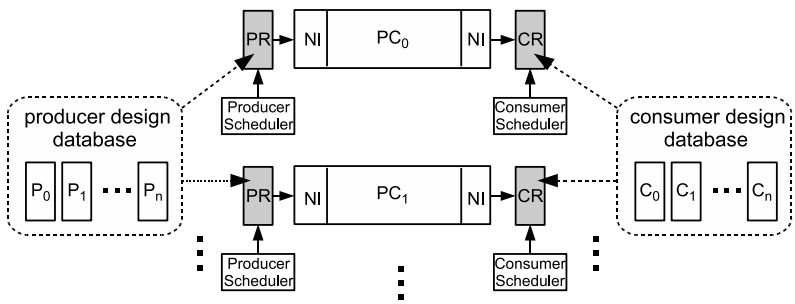
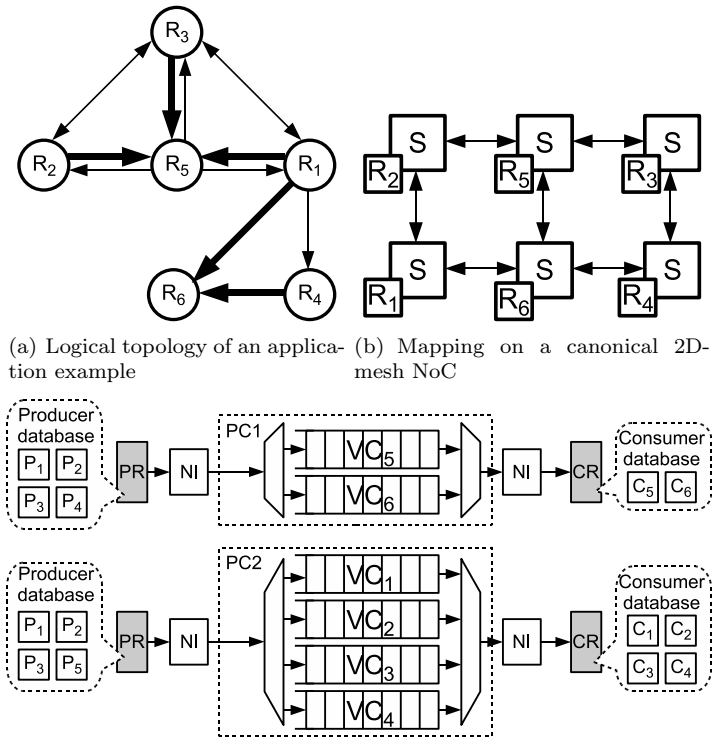


Figure 7.5. Duplicated PCs for increasing communication bandwidth

One extreme case of duplicating PCs is to equalize the number of PCs to the one of producers or consumers. Therefore we may dedicate a complete PC to each producer or consumer. To dedicate a PC to a producer (static producer in that PC) will completely remove the producer reconfiguration overhead. This dedication pattern is well suited to realize the "one-to-many" traffic, in which each producer equally transmits packets to all the consumers; on the other hand, the PC dedicated to each consumer (static consumer) will completely remove the consumer reconfiguration overhead. The resource consumption of VC buffers can be significantly reduced as well, because each PC contains only one single VC for a single consumer. This architecture is ideal for implementing "many-to-one", in which all the producers send packets to a consumer. In the extreme cases described above, more than linear bandwidth scaling can be



(a) Logical topology of an application example

(b) Mapping on a canonical 2D-mesh NoC

(c) Grouping the communications in RL-NoC for more-than-linearly performance scaling and less-than-linearly resource utilization increment

Figure 7.6. An application example of grouping communications in RL-NoC

clearly derived considering the completely removed reconfiguration overhead as well as the scheduling need for the static producer or consumer.

For each single PC in RL-NoC, packets are delivered through flit FIFOs in the same order as they are sent. However like many packet-switching networks, in-order delivery to the consumer cannot be guaranteed by the network itself in duplicated PCs [123]. Hence node designs should coordinate with the reconfiguration scheduler to reassemble out-of-order packets and synchronize the multiple PCs in use. An example solution is demonstrated in Figure 7.7: Packets to be routed in sequence to Consumer C_i are distributed in two PCs, with their delivery order messed up. C_i reads out its packets in one VC, and mean-

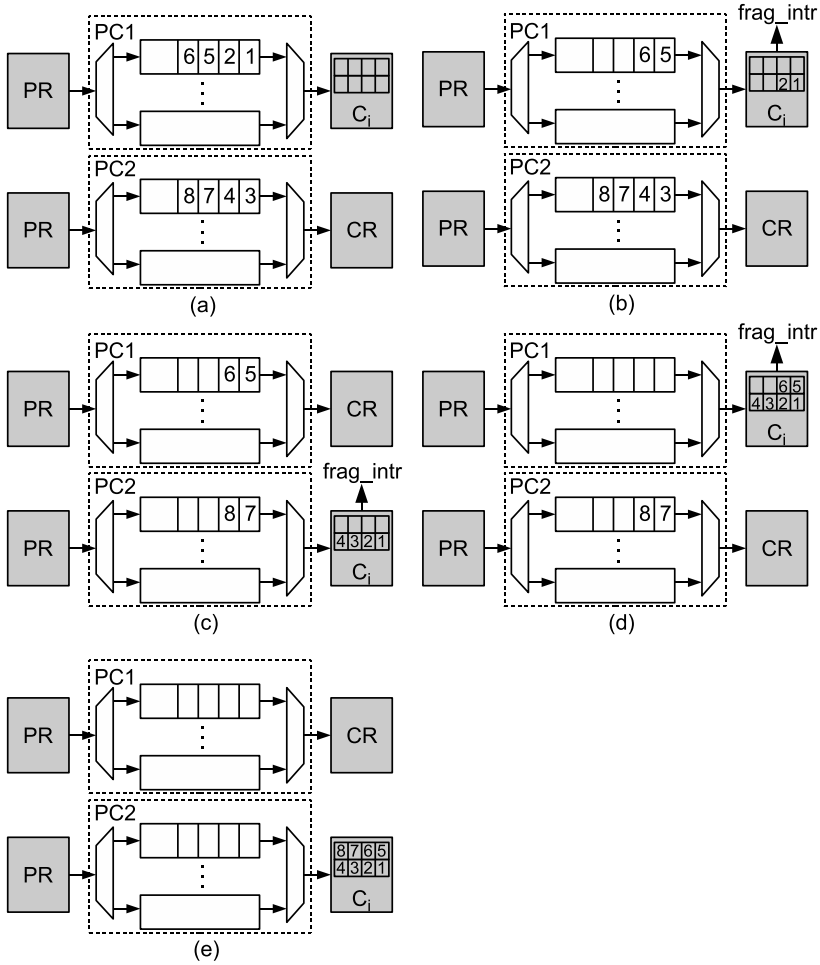


Figure 7.7. In-order packet delivery in duplicated PCs. In case of packet flow fragmentation, the active consumer node stops reading the current VC and raises an interrupt to the scheduler. It voluntarily gives up the utilization of the consumer region and waits to retrieve in-order packets from other PCs.

while spies the serial number of the next packet in this VC. In case of packet flow fragmentation (discontinuous serial number), C_i stops reading and raises a fragmentation interrupt (frag_intr) to the consumer scheduler. It voluntarily gives up the utilization of the consumer region to other consumers, and waits to be loaded in other PCs for retrieving continuous packet flow. Therefore, segmented packets in multiple PCs can be collected by the consumer in the correct order.

7.5 Implementation Results

We compare the practical hardware implementation of the proposed RL-NoC with a 2D-mesh WormHole-switching NoC (WH-NoC). Both designs are coded in synthesizable VHDL, and share a data width of 32 bits for fair comparison. The wormhole router infrastructure (see Figure 7.2) is described in detail in [43] and popularly used in many other NoC projects such as [124] and [116]. We observe in [43] its maximum clock frequencies up to 396 MHz for the controlpath and 198 MHz for the datapath when using UMC18 (180-nm) technology. To investigate the implementation on FPGAs, the wormhole router design is ported on a Xilinx Virtex-4 FX60 FPGA (-11 speed level). The timing report after implementation reveals a fastest clock frequency pair of 66/33 MHz for controlpath/datapath. On the same FPGA chip, RL-NoC can be clocked up to 222 MHz (single clock domain), which is 6.7 times faster than the wormhole router. The timing improvement is understandable: The reconfigurable RL-NoC has a much simpler structure than the wormhole router.

Table 7.1 lists the resource utilization of both designs. The first column corresponds to a single wormhole router. To construct a 4×4 homogeneous 2D-mesh, 16 routers are needed (the second column). In the router design, slice registers are utilized to implement flit buffers distributed in all input and admission channels. The last column lists the RL-NoC design. We see that a complete network architecture capable of realizing all kinds of logical interconnections of 16 nodes consumes only a small amount of LUTs and slice registers, even smaller than a single wormhole router. Intrinsic mechanism differences lead to a light-weight resource consumption of RL-NoC: It simply employs VC buffers to deliver packets, and removes the complex logic required by WH-router for flow control, routing, arbitration, lane allocation, etc. The performance/resource tradeoff can be flexibly adjusted in RL-NoC by duplicating various numbers of PCs according to application requirements. Given that 16 copies of PCs are adopted, the resource saving of RL-NoC is still considerable compared to the 4×4 WH-NoC. In the RL-NoC results, the

size-configurable flit FIFOs are implemented with Block RAMs on the FPGA. It is also feasible to migrate them into off-chip memories such as SRAMs or DDR SDRAMs in order to save the precious on-chip memory resource. In the listed statistics in Table 7.1, resource consumption of nodes are excluded for both WH-NoC and RL-NoC.

	WH-router	WH-NoC (4×4 mesh)	RL-NoC (1 PC for 16 nodes)
4-LUTs	19721	19721 × 16	2976
slice registers	4927	4927 × 16	1649
Block RAM	0	0	32 (1K depth flit FIFO)

Table 7.1. Resource utilization comparison of WH-NoC and RL-NoC

7.6 Performance Measurements

7.6.1 Experimental Setup

Cycle-accurate simulation using Modelsim has been conducted to compare the performance of RL-NoC with the canonical WH-NoC. We exclude network warm-up and cool-down cycles out of measurements and only take into account the steady state. To make the comparison meaningful, both RL-NoC and WH-NoC feature the same data width of 32-bits and run at a same datapath clock frequency. The wormhole router features 4 VCs on each PC, and adopts the X-Y routing mechanism to avoid deadlocks. In RL-NoC, the throughput-aware scheduling policy (discussed in Section 7.4.2) is chosen to maximize the packet delivery throughput for large-scale streaming applications (e.g. our targeted data acquisition and trigger systems in particle physics experiments [34]). Particularly, the consumer node will only be switched on when its dedicated VC buffer is almost full. It keeps working until all the buffered flits are read out. All producer nodes are alternately loaded to generate packets for an equivalent time slice. The time overhead for each node reconfiguration is assumed to be 10 μs, taking into account the practical ICAP designs in [31], [66] and [67] which accomplish dynamic reconfiguration in the order of magnitude of microseconds. The measurements concern various network sizes, flit FIFO sizes for RL-NoC and traffic patterns. The experimental setup for all the tests is summarized in Table 7.2 for quick reference.

The packet structure used in the experiments is demonstrated in Figure 7.8. For WH-NoC, each flit needs a domain VC_ID for VC selection in the input channel of routers. It can be dedicated to payload data in RL-NoC, leading to a higher payload transmission efficiency.

	Test 1			Test 2					Test 3
	Test 1.1	Test 1.2	Test 1.3	Test 2.1	Test 2.2	Test 2.3	Test 2.4	Test 2.5	Test 3.1
flit FIFO depth (RL-NoC)	1K			2K	1K	512	1K	1K	1K
VCF (RL-NoC)	no			no			2 VCFs	no	no
reconf. time (RL-NoC)	10 μs			10 μs				0	10 μs
network size	2×2	4×4	6×6	4×4					4×4
traffic pattern	random			random					hotspot
packet size	8 flits								
VC No. per PC (WH-NoC)	4								
VC buffer depth (WH-NoC)	4								

Table 7.2. Experimental setup of performance measurements

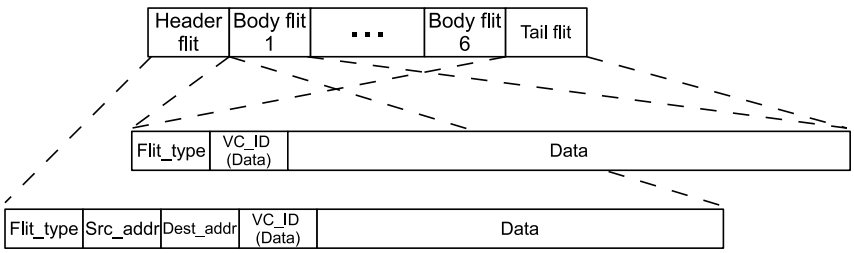


Figure 7.8. Packet structure of WH-NoC and RL-NoC

7.6.2 Results

In Test 1, we firstly study RL-NoC for various network sizes, with middle-size flit FIFOs of 1K depth for each lane. The traffic is in a random pattern, meaning that all the nodes transmite data packets to a random destination except itself. Figure 7.9 shows the packet delivery throughput results compared to corresponding WH-NoCs in 2 × 2, 4 × 4 and 6 × 6 2D-meshes. Due to the fact that all the producers are alternately multiplexed in the single node region of PR in single-PC RL-NoCs, their effective data injection rate is in fact only $\frac{1}{i \times i}$ (i = 2, 4, or 6) of WH-NoCs, as well as the throughput (red curves) before the network saturation. Hence in order to make the comparison straightforward, we assume also $i \times i$ PCs in RL-NoC corresponding to $i \times i$ routers and nodes in WH-NoC. The overall performance of duplicated PCs will be roughly estimated by the one of a single PC timing $i \times i$. We observe in the figure that blue curves for the scaled performance of RL-NoCs are comparable

with WH-NoCs (black curves). Although the saturated throughput of RL-NoC is lower than WH-NoC in the network sizes of 2×2 and 4×4, RL-NoC performs better in the 6×6 network. For different numbers of nodes occupying the reconfigurable node region in turn, RL-NoC features a comparatively stable capability to deliver packets until it is saturated. RL-NoC does not severely deteriorate the performance in large network sizes, as nevertheless WH-NoC does due to the increased traffic congestion in the random traffic pattern. The slight performance difference comes from the variation of the dynamic reconfiguration overhead with various node counts. This feature makes RL-NoC easily and efficiently scalable when more nodes are to be incorporated in the network.

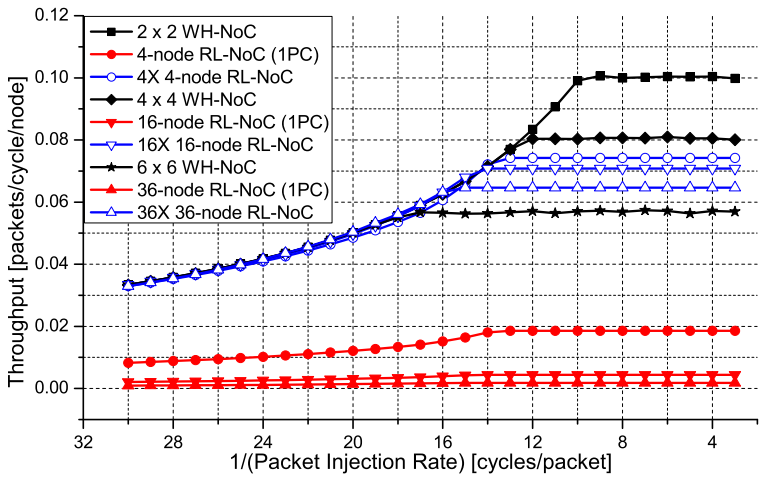


Figure 7.9. Throughput comparison of various network sizes

With a given reconfiguration throughput of the ICAP design, the overall dynamic reconfiguration overhead is dependent on the switching frequency of IP cores and directly affects the system performance. If the overhead is minimized, most of the time can be dedicated to effective data processing, therefore leading to an efficient resource utilization. In the producer-consumer reconfigurable model, the FIFO size determines the switching frequency of consumer cores: The larger the FIFO is to buffer data, the more time a consumer core can keep working for, and thus the less reconfiguration overhead it introduces. To investigate the effect of the flit FIFO size in RL-NoC which is equal to the

readout burst size for consumer nodes, we conducted measurements in Test 2 on 2K, 1K and 512 FIFO depth per lane for the 4 × 4 network. As shown with blue curves in Figure 7.10, we observe that a larger FIFO depth can alleviate dynamic reconfiguration overhead and obtain higher packet delivery throughput. If we completely ignore the reconfiguration time overhead, the network saturation will be promoted to a higher packet injection rate and therefore higher throughput (black curve). This is the physical limitation of performance which may be approached by reducing the overhead. For instance as represented by the red curve, we adopted two Virtual ConFigurations (VCF) on the consumer region to partly hide the reconfiguration time. The measured throughput is significantly enhanced and close to the ideal case without any reconfiguration overhead.

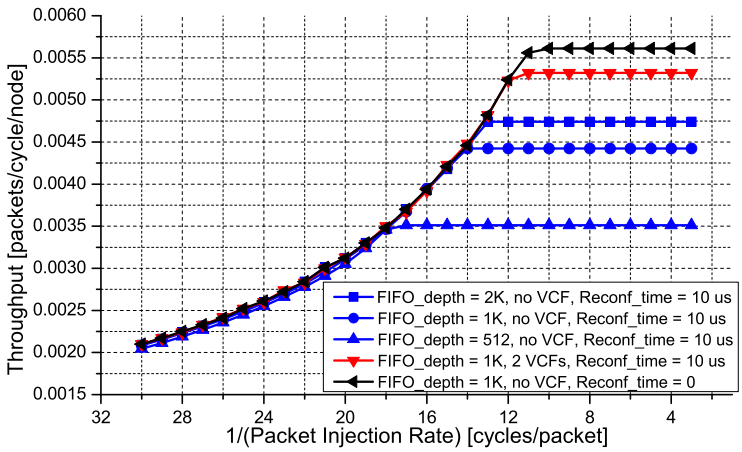


Figure 7.10. Throughput of various flit FIFO depths in RL-NoC

In addition to the random traffic pattern, we did Test 3 to compare RL-NoC with WH-NoC in a hotspot traffic pattern. The hotspot traffic often appears in the client-server communication mode, in which all other nodes (clients) talk to one node (server) and this node responds to all the others. In a baseline 4 × 4 mesh WH-NoC, the server is mapped on one of the four center nodes. All other nodes send packets to it, very easily leading the four direction links of the server node to saturation. Network congestion significantly deteriorates the packet delivery throughput of the server node. On the contrary, the uni-destination traffic pattern becomes an advantage in RL-NoC,

because the consumer region will be dominated by the server node for most of the time and the reconfiguration overhead can be largely reduced. As shown in Figure 7.11, a single PC (red curve) in RL-NoC achieves about 1/16 packet throughput on the server node as WH-NoC before the network saturation, due to the time-multiplexing packet injection of the 16 nodes. Thus 16X duplicated PCs (blue curve) achieve a similar throughput performance as WH-NoC. However the wormhole network is easily saturated after the packet injection rate is increased to around 1 packet per 120 cycles. By contrast, RL-NoC defers the channel saturation to around 1 packet per 5 cycles and enhances the saturated throughput to 12.2 times higher than WH-NoC.

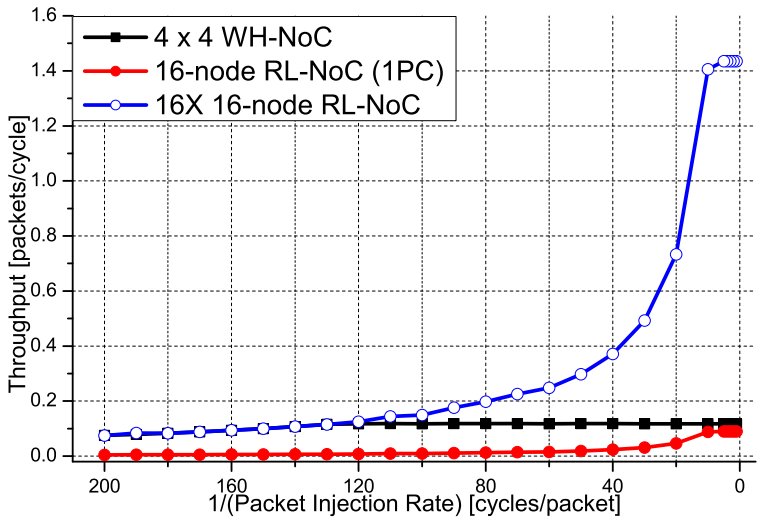


Figure 7.11. Throughput comparison in the hotspot traffic pattern

In all the above measurements, RL-NoC is assumed to run at the same datapath clock frequency as WH-NoC. Taking into account the maximum clock frequency of 6.7 times faster than WH-NoC (see Section 7.5), the RL-NoC performance can be further linearly improved by scaling the system clock frequency.

The packet latency comparison of the 4 × 4 WH-NoC and the 1K-FIFO 16-node RL-NoC is demonstrated in Figure 7.12 for both the random and the hotspot traffic pattern. We discuss the average value of a certain amount of

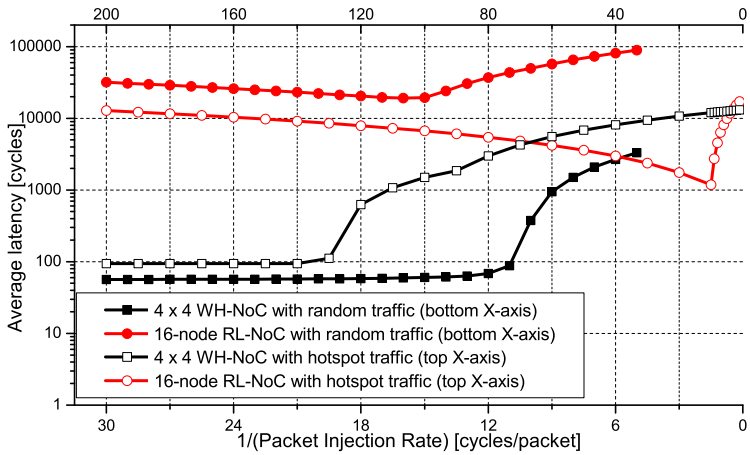


Figure 7.12. Latency comparison in the random/hotspot traffic pattern

packets. The latency is calculated from the instant when the packet is injected into the source queue to that when the packet is received by the destination node. It consists of two components: the queuing time in the source queue and the network delivery time. We observe from the figure that RL-NoC has a two to three orders of magnitude worse latency performance than WH-NoC in the random traffic pattern (curves with solid symbols). WH-NoC only significantly increases the latency after the network saturation when channel congestion happens and packets have to wait for much longer time in the source queue. In RL-NoC, various producers and consumers operate in sequence. Injected packets must wait in flit FIFOs until their destination node is configured to read them out in a burst mode. The scheduling policy which is supposed to reduce the run-time reconfiguration overhead actually worsen the latency performance. Before the channel saturation in RL-NoC, the delay is mostly from the time waiting in flit FIFOs. Therefore faster packet injection speeds up the configuration loading of consumer nodes and reduces the packet wait time in VC lanes before the readout, resulting in the descending curve segment as shown in the figure. After the channel is saturated, packets have also to wait in the source queue, leading their latency obviously increased. By contrast in the hotspot traffic pattern (curves with hollow symbols), the WH-NoC latency is deteriorated due to the heavy channel congestion after the network saturation.

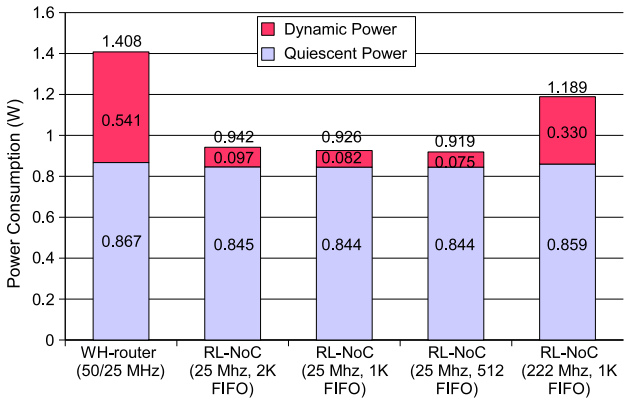


Figure 7.13. Power consumption of WH-router and RL-NoC

However the RL-NoC latency becomes lower, since most packets are destined to a same node and need not wait for that long time in flit FIFOs. The reduced reconfiguration overhead improves the latency performance of RL-NoC in the hotspot traffic pattern.

7.7 Power Analysis

We use Xilinx XPower Analyzer to compare the power consumption of WH-NoC and RL-NoC[1]. In order to estimate the power as accurately as possible, Modelsim is used to generate the signal switching activity file (VCD file) in the simulation of random-traffic experiments. The power analysis has been conducted in the context of a 4×4 WH-NoC and a 16-node RL-NoC, both working at the peak packet injection rate of 1 packet/cycle/node to maximumly saturate the network. The quiescent and the dynamic power are respectively recorded in Figure 7.13. As demonstrated by the first column, a central wormhole router in the mesh is estimated to consume 1.386 W at a

[1]At present the software tool does not support power estimation of run-time reconfigurable designs on FPGAs. However according to the experiments in [126], the authors reported an extra power consumption of partial reconfiguration in the order of magnitude of mW. In our design, the throughput-aware scheduling policy infrequently enables module reconfiguration, and hence makes the average reconfiguration power consumption even tiny in the long term system run. Therefore we neglect the power overhead introduced by run-time reconfiguration in RL-NoC.

control/data clock frequency pair of 50/25 MHz. By contrast, RL-NoC with one PC consumes much less power at the same datapath clock frequency of 25 MHz, as shown in the second, third and fourth columns. Results reveal the power reduction of 32.0%, 33.1% and 33.7% for RL-NoC with 2K, 1K, and 512 FIFO depth respectively. Even at an extreme clock frequency of 222 MHz, RL-NoC still consumes 15.6% less power than the wormhole router, as listed in the last column in the figure.

In the above power analysis, we take only into account the backbone components (router for WH-NoC and PC for RL-NoC) of the system interconnections. In the system level, RL-NoC may gain further benefits over router-based NoCs by removing the energy-consuming inter-router long wires with mutually independent PCs. This can only be quantified with future FPGAs, which must be large enough to accommodate the entire 2D-mesh WH-NoC with many interconnected routers.

"Be happy. It's one way of being wise."

Sidonie Gabrielle Colette
(French novelist, 1873 - 1954 A.D.)

Chapter 8

Conclusion and Open Issues

In this chapter, we conclude the thesis work. We summarize the advantages of the proposed design framework for adaptive computing, in comparison with the conventional static development approach on FPGAs. In addition, open issues and design challenges are listed as well to inspire the directions of subsequent researches in this area.

8.1 Conclusion

In the latest two decades, FPGA has been evolved from simply acting as programmable glue logics of ASICs into a genuine platform device containing complete system designs. As its continuously increasing market share has been seen, more and more engineers are taking the advantage of flexible programmability and hardware-level performance acceleration of FPGAs in their designs. Corresponding to the temporal parallelism of the multi-processing technology on GPCPUs, traditionally the spatial parallelism is exploited in FPGA designs. However along with the rapidly growing design complexity, new requirements arise in terms of self-adaptivity as well as efficient run-time resource management. Analogous to the software world in which multi-core technologies have been and are being popularly investigated to add spatial parallelism, modern FPGA run-time reconfigurability explores the time-multiplexity to meet new design challenges. Hence based on the FPGA run-time reconfigurability, we started the research on adaptive computing with self-awareness and led to this doctoral thesis.

We present a comprehensive and practical design framework for adaptive computing. Several design key issues are attributed into different hardware/software layers: In the hardware layer, system designs are regulated with application-specific tasks adaptively loaded into preserved reconfigurable slots residing in a general computer architecture. With respect to the run-time reconfiguration technical support, dynamic reconfiguration overhead is minimized by optimizing the ICAP designs and employing virtual configurations. In order to manage the reconfigurable system, scheduler is located in the highest-level application software layer for easy implementation and portability. The scheduler program detects trigger conditions and optimally swap hardware tasks, with the help from device drivers and the OS. In case of module swapping, hardware context switching happens and mechanisms of saving and restoring the context may be required. In addition, inter-process communications may exist among hardware modules. We analyze the characteristics of one solution named pipe and demonstrate its hardware implementations.

To verify the proposed design framework, we apply the self-adaptation concept to practical embedded designs. In the first case study, a NOR flash memory controller and an SRAM controller time-share the same programmable resources on the FPGA. By means of being scheduled in an appropriate order, both controller modules can be alternately loaded into the reconfigurable slot, according to different memory access requirements on either the flash or the SRAM. The adaptive design reveals less resource consumption in comparison with statically placing both controllers in the system. The required functionalities are entirely accomplished, including booting the system from the flash memory and addressing the SRAM for application-specific computation.

In order to investigate our target application in nuclear and particle physics experiments, we model the correlated multi-stream processing and compare the adaptive architecture with the conventional static approach. Based on the model analysis, a system design is constructed consisting of two real particle recognition computing engines. Experimental results not only demonstrate the expected benefits obtained by automatically adapting different algorithm engines in a same reconfigurable slot, but also expose some design challenges for which further studies might head.

We then put the FPGA run-time reconfiguration technology in the scenario of on-chip communications. A routerless interconnection architecture is presented on the basis of the adaptive design framework. Significantly distinguished from the existing interconnection architectures, such as buses, crossbars, or NoCs, the so-called RL-NoC takes advantage of the time-multiplexingly point-to-point communication infrastructure with packet injection and retrieval decoupled. Compared to the present hot topic of NoCs, the routerless archi-

tecture features lower design complexity, less resource consumption, higher work frequency as well as more efficient power dissipation. RL-NoC is observed to achieve comparable or even superior packet delivery efficiency of a wormhole NoC in the experiments. It is regarded as a promising alternative of canonical NoCs in some design scenarios on FPGAs, especially in light-weight applications.

According to the analysis and experimental results, we may partly conclude the merits of self-adaptive designs in the following items:

- **Easy design management.** Traditionally design components are managed and incorporated in the system by designers during their development period. The static and offline resource management is complex and error prone, especially in massive processing systems with hundreds of FPGAs. By contrast with our proposed framework for self-aware autonomic designs, the base system architecture for all FPGA chips becomes uniform by reserving PR slots as blackboxes. According to run-time trigger conditions, the scheduler decides the most urgent task and dynamically allocates computation resources. Adapting reconfigurable modules on the fly not only simplifies the design management work of designers, but also makes it more flexible and accurate.

- **Reduced hardware cost.** By dynamically loading different modules, multitasking is realized within the same reconfigurable region. This feature is especially well suited for the cases in which some functionalities are only occasionally or exclusively required. As payoff, the FPGA chip capacity or count requirements are alleviated, which should be otherwise large enough to contain all functional modules in conventional static designs. The hardware platform may be retained if more functional modules are to be added in the new product design. Moreover, PCB designs may also be simplified by migrating inter-chip bandwidth into inter-process communications within an FPGA.

- **More efficient utilization of computing resources.** Multitasking in PR regions reduces the resource requirement for the same set of expected functionalities. In addition, flexible scheduling policies activate in turn different modules according to certain conditions. FPGA resources are less frequently kept idle and more efficiently utilized to accomplish high performance/cost ratio. Energy consumption is also foreseen to be reduced by removing unnecessary static placements in the system.

- **Advanced features.** The self-awareness using the scheduler program to monitor the system, is the basis of several advanced technologies such

as self-evolving or self-healing [127] [128] [129]. More intelligent systems can be developed under this adaptive framework.

Meanwhile in the process of our research, we also see some issues or challenges on which special attention should be paid when applying this technology to practical designs. The aspects mainly include:

- **High throughput vs. low latency.** Low latency packet delivery to PR modules is a main challenge of run-time reconfigurable designs, because it is divergent with reducing the reconfiguration overhead for high throughput by minimizing the module switching frequency. No matter whether being the raw data or the IPC information between modules, data packets have to wait in a buffer device until their destination module is scheduled and activated in the shared reconfigurable slot. In order to realize quick response with real-time requirements, the corresponding module is to be immediately configured to react. This scheduling policy will inevitably deteriorate the run-time reconfiguration overhead and do harm to the overall throughput performance of the system.

- **Fragmentation problem of module placement.** When fitting modules in a preserved PR region, not all modules will perfectly occupy all the resources. The unused resources will be kept idle and lead to inefficient utilization. This phenomenon is known as *internal fragmentation*, and has been reflected in our second case study as an example. In principle, various modular designs with similar sizes can be more efficiently fitted in the same reconfigurable slot. Moreover more module counts sharing the reconfigurable slot can alleviate this problem to some extent as well.

8.2 Open Issues for Future Work

In this project, some issues remain still open due to the time reason and limited manpower. We may list some of them as inspiration to the researchers engaged in this field in their future work. The directions include:

- **Design automation in software tools.** With respect to run-time reconfigurable designs and online resource management, most of the design loops are still done manually at present. Design automation in software tools is expected to improve development productivity and ease the designers.

- **Low latency packet delivery for real-time applications.** As we have discussed ahead, low latency packet delivery is not straightforwardly achievable because of the intrinsic mechanism of reconfigurable designs. To address this issue, one promising research direction might be to implement the Quality-of-Service (QoS) support in the scheduler to provide prioritized traffics. Data packets with real-time requirement may monopolize dedicated lanes and be digested with latency awareness. Others are delivered under the discipline of minimizing the reconfiguration overhead. Through prioritizing the traffic, both the latency and the bandwidth requirements might be simultaneously met.

- **Fast reconfiguration speed and low reconfiguration overhead.** Currently the dynamic reconfiguration time overhead is estimated in the order of magnitude from microseconds to milliseconds, depending on the module size. Wider configuration bandwidth or other technologies are expected to further minimize the reconfiguration overhead. We may foresee that reconfigurable modules will be allowed to be more frequently swapped as the reconfiguration overhead decreases. This situation will certainly enhance the real-time performance of adaptive systems by immediately equipping the corresponding functional modules.

- **Fine-grained module placement.** Concerning the aforementioned internal fragmentation problem, fine-grained module placement [130] [131] is being under investigation to alleviate the resource waste problem in run-time reconfigurable designs.

- **Massive reconfigurable slot management.** As the granularity of reconfigurable slots decreases and accordingly their number increases, the management and scheduling issue will become more complicated from 1D to 2D. In that case, how to allocate and combine several fine-grained regions to fit a complete module design will be a challenging scheme.

- **Online module placement and routing.** As the performance advance of host processors in embedded systems, it might turn into the reality to dynamically conduct design placement and routing on the fly. This technology will contribute to flexibly locate design modules in different positions of any shape on the FPGA, without having to implement all the module combinations offline.

Appendix A

Design and Development of ATCA-based Compute Node

To manage the large data rate from detectors, we construct a hierarchical network architecture which consists of interconnected Compute Nodes (CN). The connectivity is classified in two categories: external connections and internal ones. The external channels are used to communicate with detectors and the PC farm, to receive detector raw data for processing and forward results for storage and offline analysis. Specifically they provide optical and Ethernet links. The internal connections bridge all algorithms or algorithm steps for parallel/pipelined processing. Both on-board I/Os and the inter-board backplane interface function as internal links. The detailed system architecture will be introduced in the following sub-sections starting with the interconnected network and working our way down to the node design.

The themes of this chapter concern paper [32], [33], [34], [35], [36] and [37] listed in Section 1.4.

A.1 Global Computation Network

The Advanced Telecommunications Computing Architecture (AdvancedTCA or ATCA) [58] standard is architected to provide the bandwidth needed for the next generation computation platform. A full-mesh shelf backplane (see Figure A.1) can support 2.1 Tbps of data transport when using 3.125 GHz signaling and 8B/10B [132] encoding. In physics experiment applications, pattern recognition algorithms are partitioned and distributed in many compute nodes

for high processing throughput. Up to 14 nodes can be fitted in one ATCA shelf and they are mutually interconnected through the backplane. Direct P2P connections provide much flexibility for various network configurations, such as vertical pipelined processing, or horizontal parallel processing, or hybrid solutions with more complicated interconnections. This feature enhances the platform general-purpose for different applications with different network architectures. In addition it provides significant freedom and convenience to partition processing logics across multiple boards.

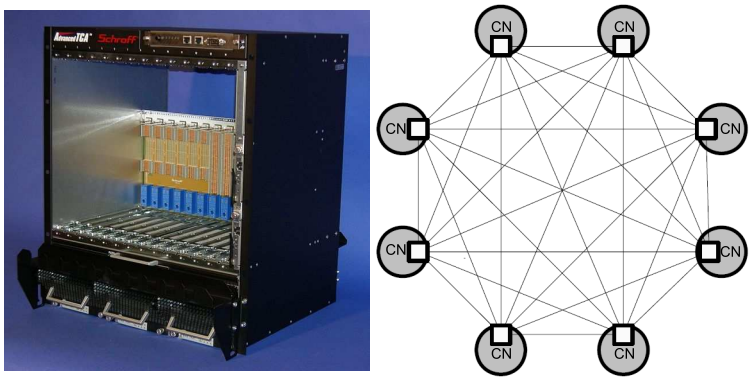


Figure A.1. ATCA crate and full-mesh backplane (only 8 nodes shown)

Figure A.2 shows the network topology in experimental facilities, where multiple ATCA crates are used to meet high communication and computation requirements. Through bonded optical channels and switches, raw data are dumped from the front-end circuits, which take care of sampling and digitizing analog signals generated by detectors. Afterwards all data will be processed in the network for pattern recognition, correlation, event building and filtering. The processing modules are partitioned and reside in FPGA cells. All the steps constitute the complete computation by communicating through the hierarchical interconnections, including on-board I/Os, inter-board shelf backplanes, and perhaps also the inter-crate optical link or Ethernet switching if necessary. On-board channels provide large bandwidth, while the inter-crate switching has more communication overheads and will introduce latency penalty. Thus trying to group the computation steps with high mutual communication requirements on the same board or next in the same crate, is a basic rule to implement the algorithms in practice. After pattern recognition and event

selection in the network, a large proportion of event data is discarded on the fly while only a small part will be labelled as interesting and forwarded to the PC farm via Ethernet for storage and in-depth offline analysis.

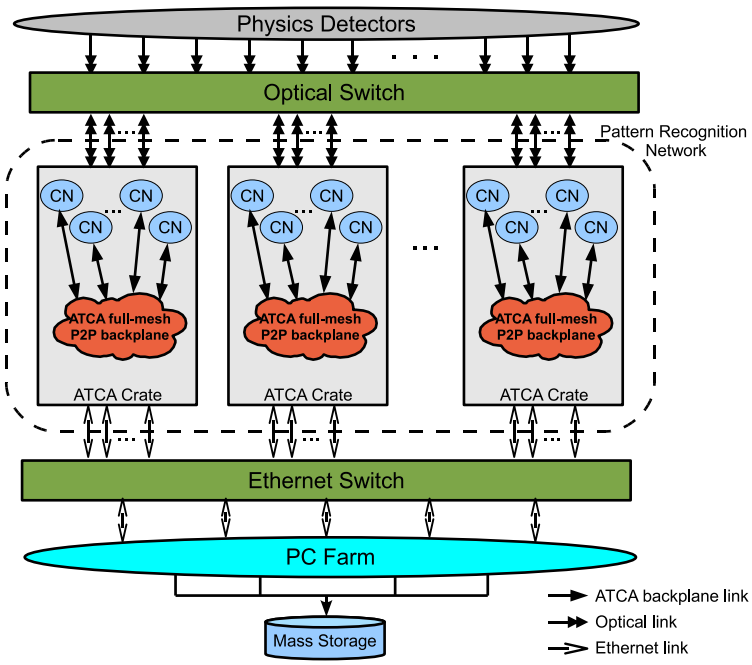


Figure A.2. Online pattern recognition network. The system features a hierarchical architecture constituted by interconnected compute nodes.

A.2 Compute Node

The system uses Xilinx platform FPGAs (with hardcore embedded processors) as primary processing components. For the first prototype Virtex-4 FX60 is chosen. On future products the up-to-date generation FPGAs might be adopted instead. Figure A.3 shows the schematic of the Compute Node (CN) board. Each board consists of five FPGAs, four of which (No. 1 to 4) work as algorithm processors and the fifth (No. 0) as a switch interfacing to other CNs through the ATCA backplane. Each processor FPGA has two

RocketIO Multi-Gigabit Transceiver (MGT) based optical links, which can run at a maximum baud rate of 6.5 Gbps per channel. In addition, all the FPGAs are equipped with one Gigabit Ethernet as well as 2 GBytes DDR2 memory each. The total 10 GBytes memory capacity is mainly for data buffering and large Look-Up Table (LUT) storage purpose.

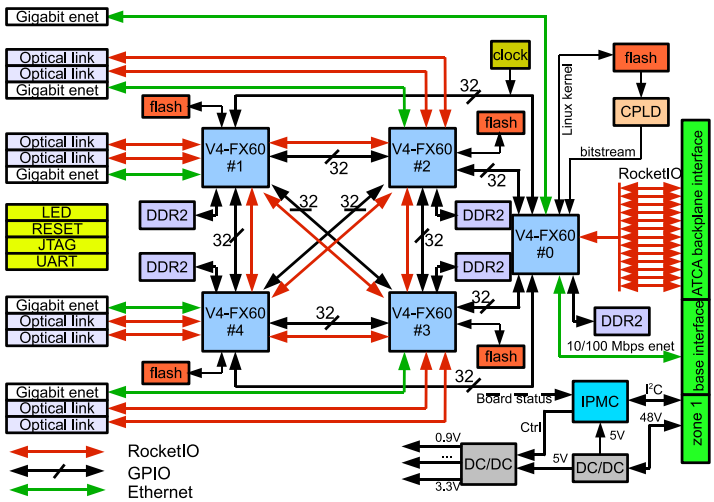


Figure A.3. Compute node schematic. Five FPGA chips are networked with on-board connections. Memory and peripheral components as well as communication links are placed on the board.

For many physics experiments in the future, their processing algorithms or partitions are still unclear and may feature different traffic patterns in the network. Thus to make the board design capable of easily porting high-performance algorithms, all four processor FPGAs are interconnected with each other in a full-mesh topology. The connectivity includes both 32-bit General-Purpose IO (GPIO) buses and one full-duplex RocketIO link per connection. These processor FPGAs also connect to the switch FPGA with dedicated 32-bit GPIOs. Either circuit-switching or packet-switching is able to be configured to communicate with other CNs in the crate. 16 RocketIO channels to the backplane feature the bandwidth of 104 Gbps at 6.5 GHz signaling. Besides the switch structure, sub-event data from all four processor FPGAs can be collected in the switch FPGA and conduct event building and filtering.

With the on-board P2P interconnections, it is convenient to partition unexpected algorithms for different experiments and aggregate all five FPGAs as a virtual one with five times capacity.

NOR flash memories are mounted on the board for Operating System (OS) kernel and FPGA bitstream storage. A customized Intelligent Platform Management Controller (IPMC) add-on card fullfills the ATCA requirements on power negotiation, voltage monitoring, temperature sensoring, and FPGA configuration check, etc. Figure A.4 shows the picture of our first prototype CN PCB and Figure A.5 shows the second version. To meet the dimension requirement, all main components are placed on the top side except five ultra low profile Small Outline Dual In-line Memory Modules (SO-DIMM) DDR2 SDRAMs.

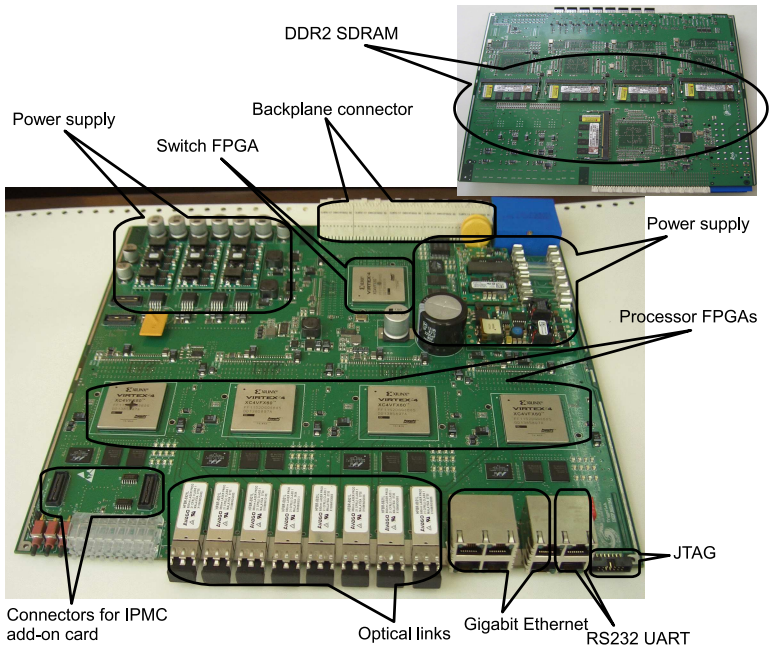


Figure A.4. Prototype PCB of the compute node

Each CN resides in one of the 14 slots in the ATCA crate. The power budget for each slot from the crate is 200 Watts at maximum, larger than the worst-case estimation of 170 Watts of the CN board. When all 14 nodes

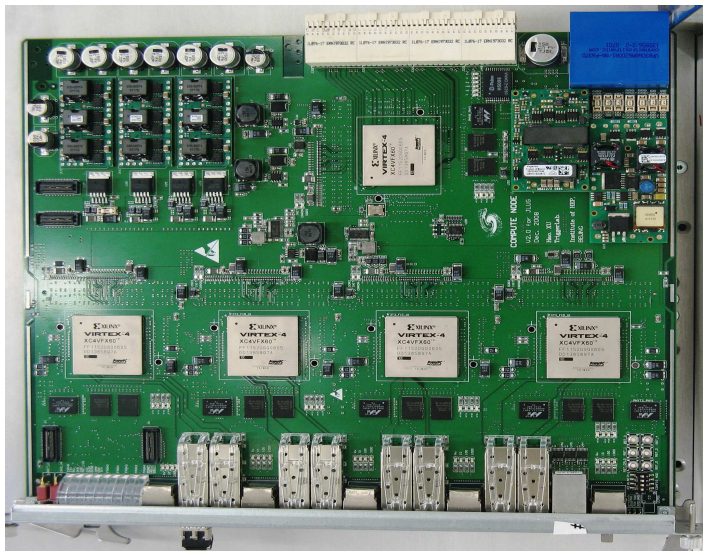


Figure A.5. Compute node PCB version 2

are pluged in, such a crate can host up to 1890 Gbps inter-FPGA on-board connections (GPIOs at 300 Mbps), 1456 Gbps inter-board backplane connections, 728 Gbps full-duplex optical bandwidth, 70 Gbps Ethernet bandwidth, 140 GBytes DDR2 SDRAM, and all computing resources of 70 Virtex-4 FX60 FPGAs.

A.3 HW/SW Co-design of the System-on-an-FPGA

A.3.1 Partitioning Strategy

The DAQ and trigger system in physics experiments request more features for convenient experiment operations than fundamental data processing. For example due to temporal and spatial limitations, operators would like to remotely and dynamically reconfigure and control the platform without approaching to the experimental facility. A friendly user interface also helps physicists to easily adjust experimental parameters and monitor the system

status. In our platform, hardcore PowerPC microprocessors on FPGAs can be utilized to implement versatile control tasks, while locating the performance-critical computation in the FPGA fabric in hardware. Concrete criteria are described as follows:

1. All pattern recognition algorithms are to be customized in the FPGA fabric as hardware co-processors, working in parallel and/or pipeline to identify interesting events.

2. Slow control tasks are implemented in software by high-level application programs which execute on top of embedded microprocessors and operating systems.

3. The integrated soft TCP/IP stack in the operating system is employed for Ethernet transmission.

A.3.2 Hardware Design

Based on the modular design approach, we develop the hardware system on the FPGA using hardcore and softcore components. As shown in Figure A.6, the PowerPC 405 processor, the Multi-Port Memory Controller (MPMC) [133], and other peripherals constitute a complete embedded computer system. Compared to the canonical bus-based design, MPMC provides direct ports to memory-hungry modules and significantly speedup memory accesses: Incoming detector sub-events are buffered in DDR2 via RocketIO-based optical wrappers, and afterwards processed by detector-specific algorithm co-processors. In this system, the PLB bus is used only for low data rate peripheral communications and controls.

For various applications of physics experiments, the system architecture is intended to be fixed and only to replace algorithm engines. It enables the design re-usability and largely shrinks development time.

A.3.3 Software Design

The open-source embedded Linux kernel of version 2.6 was ported to the PowerPC processor. The soft Linux TCP/IP stack (including UDP/IP) drives the Ethernet transceiving with commodity PC clusters. Device drivers for standard peripherals can be enabled when Linux is configured, including Tri-mode Ethernet, RS232 UART, flash memories, etc. Others for algorithm modules must be customized. Based on the operating system and software development kits, flexible applications can be exploited varing from C/C++

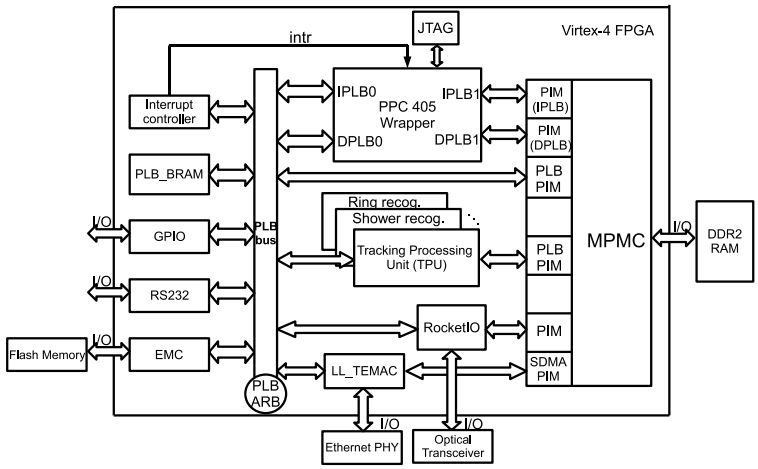


Figure A.6. MPMC-based hardware design. In addition to general components, customized algorithm engines are incorporated in the system for application-specific computation.

or Java programs to high level scripts. Programs running on PowerPC processors mainly offer user friendly interfaces for system monitoring and parameter adjustment, drive TCP/IP communications with PC farms, and also assist hardware for co-processing.

One main feature of the software design is its zero budget. Components including the OS, the file system generator, the cross-compilation tools, and some benchmark programs all come from the open-source community.

Appendix B

Implementation of Particle Recognition Algorithms

In our project timeline, HADES is the first experiment in using the computation platform to upgrade the existing DAQ and trigger system for heavier ion reactions and higher processing requirements. The promoted particle reaction rate might reach 100 KHz, implying a maximum raw data rate up to 10 Gbytes/s. We estimate to use one single ATCA crate, two compute nodes for each detector sector and twelve for six sectors in total.

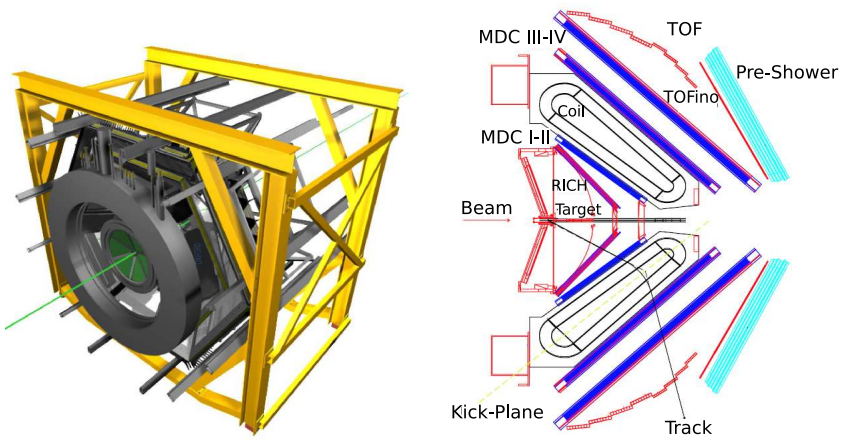
Along with the detector construction, we have been developing and evaluating particle recognition algorithms on our reconfigurable platform, specifically MDC particle track reconstruction (for MDC detectors) and Cherenkov ring recognition (for the RICH detector). Both algorithm processors receive the readout raw data and search for certain patterns. Their processing results are also correlated for precisely identifying interesting physics events happened on detectors. Only interesting events are assembled and forwarded to the mass storage. The noise data are discarded on the fly.

The content of this chapter concerns paper [38], [39], [40], and [41] listed in Section 1.4.

B.1 Track Reconstruction in MDCs

In high-energy physics experiments, the momenta of charged particles are studied by observing their deflection in magnetic field. The so-called Mini Drift Chamber (MDC) detectors are used to reconstruct the particle tracks

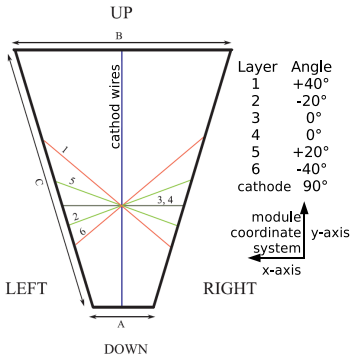
entering and leaving the magnetic field, for further deriving the deflection angle inside it. The HADES tracking system consists of four MDC modules which have six identical trapezoidal sectors (see Figure 2.1 for dismounted and B.1(a) for mounted view). Two MDC layers are located before and two behind the toroidal magnetic field which is produced by 6 superconducting coils, as illustrated in Figure B.1(b). In first approximation the magnetic field does not penetrate into the MDCs. Thus particle tracks only bend in the magnetic field and the segments before or behind the coil could be approximately described by straight lines. The two segments can be reconstructed separately with the inner (I - II) and the outer (III - IV) MDC information. The basic principle is similar and hence in this article we focus only on the inner part for explanation.



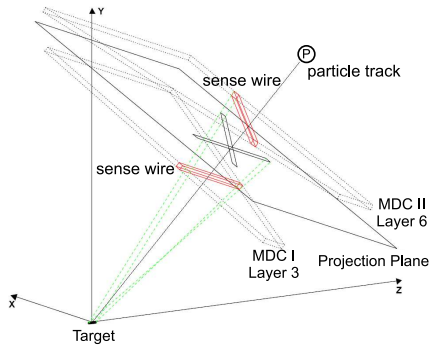
(a) Mounted view (b) Side view

Figure B.1. The HADES detector system

In the two inner MDC modules, a total number of 12660 sense wires (6 sectors) are arranged in 12 layers and 6 orientations: +40°, -20°, 0°, 0°, +20°, -40°, shown in Figure B.2(a) for one trapezoidal sector. When the beam hit the target, charged particles are emitted from the target position and go forward through different wire layers in straight paths. Along their flying ways, pulse signals are generated on sense wires close to the tracks with high probability (>95%). We also say that the sense wires are "fired" by flying particles. Shown in the coordinate system in Figure B.2(b), if the sensitive volumn of each wire is projected from the boundary of the target onto a plane located between two inner chambers, apparently the particle passed through the projection plane

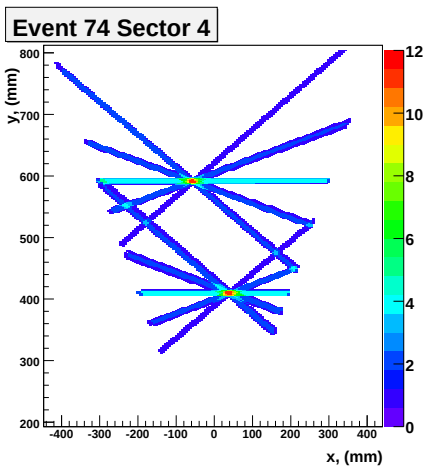


(a) One sector of the MDC detector with six orientation wires

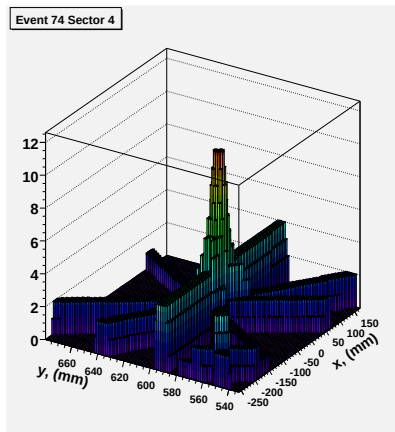


(b) Track reconstruction in inner MDCs. The track penetration point on the projection plane can be identified from the projection overlap of fired wires in different layers.

Figure B.2. Particle track reconstruction in HADES MDCs



(a) Projection plane with two passed particles. The color shows the number of overlapped wire projections on each pixel.



(b) 3D display of projection accumulation for a single track. The peak in the center specifies the penetration point on the projection plane.

Figure B.3. Track penetration points on the projection plane

at the point where all projections of fired wires from different layers overlap. To search for such regions, the projection plane is treated as a two dimensional histogram with the projection area as bins (pixels). For each fired sense wire, all the pixels covered by its projection are increased by one. By finding the locally maximum pixels whose values are also above a given threshold, track points can be recognized and the tracks are reconstructed as straight lines from the point-like target to those peak pixels. Figure B.3(a) demonstrates the 2D projection plane for one sector with two penetrating particles. We observe that the projection of fired wires from the total 12 layers overlap in two dots, which feature the amplitude peaks and represent two particle tracks. Figure B.3(b) is the 3D display of Figure B.3(a) for a single track, where the coordinates of the peak in the center is recognized as a track point.

An offline built Look-Up Table (LUT) is employed to tell which pixels on the projection plane are touched by the projection of every fired sense wire. Thus realtime coordinate calculation can be avoided considering its geometrical complexity for FPGA implementation. In practice, we choose the resolution of 128 $\times$ 256 pixels for each sector. The *projection_LUT* is about 1.5 MBytes per sector and is feasible to be initialized in the DDR2 memory.

B.2 Ring Recognition in RICH

The HADES Ring Image CHerenkov (RICH) detector is used to identify dilepton pairs based on the Nobel Prize winning discovery of Cherenkov effect. As explained in [92], a charged particle emits light cone (Cherenkov radiation) when it travels through a transparent substance with a speed faster than the speed of light in that material. Specifically in the HADES experiment, dileptons are emitted from the collision. They fly at a high velocity through the inner MDC detectors from the target. Therefore the generated Cherenkov light cone can be reflected by the mirror and displayed on the RICH detector in the shape of a ring. The ring pattern is searched on the RICH plane with a resolution of 96 $\times$ 96 pixels per sector. According to the physics principle, the Cherenkov ring from the dilepton pair features a constant diameter equivalent to the distance of 8 pixels on the RICH plane. As shown in Figure B.4, the ring pattern search is conducted within a fixed mask region of 13 $\times$ 13 pixels for each potential ring center. The hits on a ring with a radius of 4 pixels are added to the value *ring_region*. There are two veto regions inside or outside the ring region, where hit pixel counts are also accumulated. The ring pattern can be identified only if both the *ring_region* sum is above and the *veto_region*

sums are below their respective thresholds. The thresholds are programmable during the experiment.

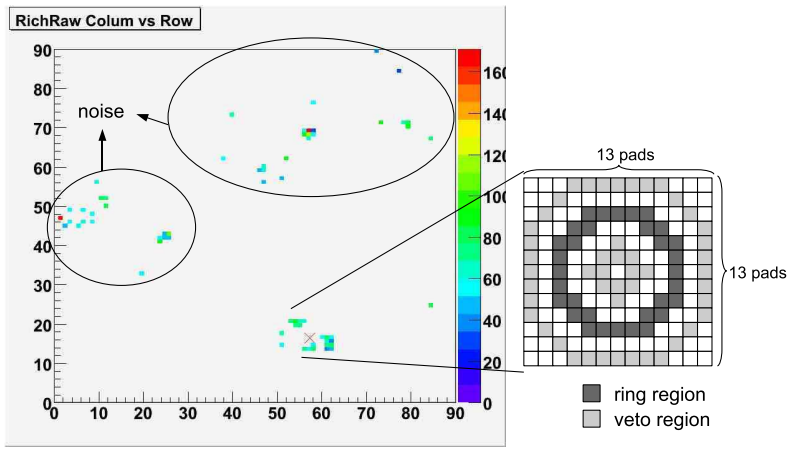


Figure B.4. Fixed-diameter ring recognition on the RICH detector

Because of the constant diameter of ring patterns, the computation challenge falls on position identification of ring centers. In the original design of [134] [135] from J. Lehnert et al., they treat all the 96 × 96 pixels on the RICH plane as potential ring centers: With the received RICH sub-events containing the position of all hit pixels from the detector readout circuits, the complete hit information of the RICH plane is reconstructed in a memory device. Afterwards ring patterns are searched within respective mask regions of all the pixels as ring centers, in parallel for 96 columns on 12 Xilinx FPGAs [134] [135]. To treat all the pixels as ring centers is not only computation inefficient, but also resource consuming on FPGAs. In addition, it requires extra work to correlate the RICH results with the rest detector system (especially inner MDCs) in the offline analysis. In order to simplify ring recognition and correlate the RICH pattern with the inner MDC tracking information, identified particle tracks in inner MDCs are introduced to point out potential ring centers. The particle tracks are reflected by the mirror onto the RICH plane. Hence the coordinate of the track penetration points on the MDC projection plane is converted into the one of potential ring centers on the RICH plane. To avoid complex online geometrical calculation, the coordinate conversion task is done offline to arrive at a Look-Up Table (LUT). Taking into account the coordinate conversion

error due to the resolution difference from MDC to RICH, normally we map a single particle track to multiple neighboured pixels in a search window (e.g. 5×5) on the RICH plane. Hence interesting ring patterns can be prevented from being ignored due to the slight coordinate conversion error.

With the small number of specified ring center candidates, we do not have to reconstruct the complete hit information for all the pixels on the RICH plane, but need only traverse all the hit pixels belonging to the same RICH sub-event to judge their positions. If they fall into the ring region of a center candidate, they may come from the valid Cherenkov light generated by flying dileptons; otherwise they are probably the noise to be discarded. The position judgment is realized by geometrical calculation on the distance between the hit pixel and the ring center, as demonstrated by the VHDL-syntax code in Algorithm 4.

Algorithm 4 Position judgment of a hit pixel to a ring center candidate

```
{(x1, y1) : position coordinate of the hit pixel.}
{(x0, y0) : position coordinate of the ring center candidate.}
{x_distance = |x1 - x0|; : distance on x axis from the hit pixel to the center candidate.}
{y_distance = |y1 - y0|; : distance on y axis from the hit pixel to the center candidate.}
{hop = x_distance + y_distance; : hop distance from the hit pixel to the center candidate.}

position_result = FARAWAY_NOISE_REGION; {by default the hit pixel is regarded as
noise.}

if hop ≤ 2 then
   position_result = INNER_VETO_REGION; {the hit pixel falls into inner_veto_region.}
end if

if ((hop = 5 or hop = 6) and x_distance ≤ 4 and y_distance ≤ 4) or (y_distance = 0 and
x_distance = 4) or (y_distance = 0 and x_distance = 4) then
   position_result = RING_REGION; {the hit pixel falls into ring_region.}
end if

if (hop = 9 and x_distance ≤ 6 and y_distance ≤ 6) or (y_distance = 6 and x_distance ≤
2) or (x_distance = 6 and y_distance ≤ 2) then
   position_result = OUTER_VETO_REGION; {the hit pixel falls into outer_veto_region.}
end if

return position_result;
```

After the position judgment of all the hit pixels corresponding to all the potential ring centers, the hit counts in the ring region as well as in the inner/outer veto regions can be accumulated. They will be compared to the thresholds for determining whether a ring pattern is successfully identified or

not. Only the data with identified patterns are to be retained. The noise will be discarded on the fly for reducing the data rate before storage.

B.3 Implementation

Both algorithms are implemented in the FPGA fabric as hardware processing engines. Figure B.5 elaborates the design structure of the Tracking Processing Unit (TPU) and the Ring Recognition Unit (RRU), which are to be integrated in the system design shown in Figure A.6. In the system design, event data are imported from the detector front-end circuits into the FPGA via optical links [34]. They are continuously supplied to the input FIFO residing in the slave interface of each algorithm engine. The TPU core extracts the fired wire serial numbers and derives the address information for each wire using an *address_LUT*. With the storage address of the *projection_LUT* in the DDR2 memory and the position address on the projection plane specified, a master device reads out the *projection_LUT* data for each fired wire, and feeds them to the *accumulate_unit* in which the projection overlap histogram is accumulated. Afterwards the *peak_finder* module figures out the peaks by comparing each pixel with all its eight neighbours. The comparison is conducted simultaneously on all the 128 pixels in a row for parallel processing. The complete projection plane is scanned row by row to identify peaks. Identified peak pixels represent the positions in which particle tracks penetrate on the projection plane. These results are both collected in the output FIFO for being recorded, and induced into the *ring_center_buffer* to specify potential ring centers for RICH ring recognition.

The incoming RICH sub-events contain the positions of hit pixels on the RICH plane. All the hit pixels belonging to the same event are to be extracted and buffered in the *single_event_buffer* in RRU. Meanwhile, ring center candidates are derived from particle tracks, converting their position coordinates in the MDC projection plane into the RICH plane with a LUT. Therefore ring pattern search is conducted in *ring_search_units* within the mask region of the specified ring center candidates. The number of parallel *ring_search_units* is configurable according to the available resources on the FPGA. Each *ring_search_unit* takes charge of one ring center pixel as well as its neighboured pixels in a search window, which are also deemed as center candidates for avoiding the coordinate conversion error from MDC to RICH. Specifically for a search window of 5×5, there exist in fact 25 processing cores in each *ring_search_unit* corresponding to the single derived center candidate and its 24 neighbours. Loading the ring center candidates

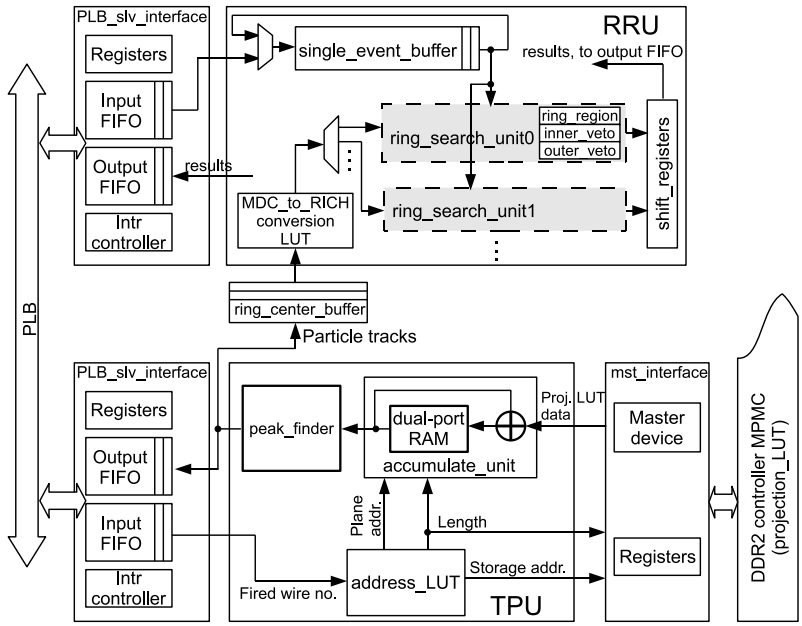


Figure B.5. Hardware design of the algorithm engines

in *ring_search_units* is also in the unit of a same event. If the number of configured *ring_search_units* is larger than or equal to the center candidate count of an event (i.e. the number of found particle tracks from MDCs), the hit pixels in *single_event_buffer* can be simply read out and traversed for deciding their positions and identifying rings from the accumulated values of *ring_region*, *inner_veto_region* and *outer_veto_region*. Otherwise ring centers have to be loaded in *ring_search_units* for multiple computation rounds, and accordingly all the hit pixels belonging to this event must be reiterated in the *single_event_buffer* until all the centers are done. After each round computation, the center candidates with recognized ring patterns are shifted out and collected in the output FIFO for result archiving.

Both TPU and RRU feature a parallel and pipelined design structure. With many processing units instantiated for histogram accumulation and peak finding in TPU, and ring pattern search in RRU, the hardware engines accelerate the application-specific computation even though they work at lower clock frequencies than General-Purpose microprocessors (GPCPU). Moreover, fine-

grained memory inside the FPGA using Block RAMs is rather efficient to implement fast memory accesses such as LUTs. This also contributes to the hardware acceleration on pattern recognition algorithms.

B.4 Results

B.4.1 Implementation Results

Resource utilizations of RRU with 1 or 2 *ring_search_unit* configurations are listed in Table B.1, as well as the TPU design. Interface blocks are included in the reported results. We observe that both TPU and RRU consume a reasonable fraction of available resources on a Xilinx Virtex-4 FX60 FPGA. Taking into account the system design which consists of embedded processor, Multi-Port Memory Controller (MPMC), peripherals plus the algorithm engines, it is feasible to implement the complete computer system on a single Virtex-4 FX60 FPGA for particle recognition computation. The resource utilization is acceptable and still enables the possibility to upgrade the system in future designs.

Resources	**RRU** (1 ring_search_unit)	**RRU** (2 ring_search_units)	**TPU**
4-input LUTs	4728 out of 50560 (9.4%)	8186 (16.2%)	6072 (12.0%)
Slice Flip-Flops	3670 out of 50560 (7.3%)	5190 (10.3%)	3815 (7.5%)
Block RAMs	31 out of 232 (13.4%)	31 (13.4%)	48 (20.7%)

Table B.1. Resource utilization of TPU and RRU

The timing reports reveal that the TPU module can run at 125 MHz maximumly. To match the speed of the PLB bus as well as the interface design, its clock frequency is chosen as 100 MHz in practice. The RRU design features two clock domains: The PLB interface runs at 100 MHz, and the RRU core can run at a frequency up to 160 MHz. These two clock domains are coupled by asynchronous input and output FIFOs.

B.4.2 Performance Estimation

Experimental measurements have been done on the TPU design with various wire multiplicities, meaning that different numbers of wires in different positions are fired for each event. A single TPU core is observed to achieve a processing speed ranging from 1.0 to 32.3 KSub-events/s, depending on the

wire multiplicity. This implies a performance speedup from 10.8 to 24.3 times than an Intel Xeon 2.4 GHz CPU core in the particle track recognition processing. Taking into account the ion reaction rate up to several tens of KHz in HADES, roughly 18 to 30 TPU cores are required for the total 6 detector sectors. With respect to the RRU design, the improved approach with specified ring centers from particle tracks can outperform the previous system described in [134] and [135] by about one hundred times, achieving a processing speed in the order of magnitude of MSub-events/s. Therefore 6 RRU cores are more than sufficient for HADES ring recognition, with one allocated for each sector. Roughly estimating, one TPU/RRU pair achieves a computation capability equivalent to several tens up to hundred of commodity PCs for HADES particle recognition. In our customized computation platform [34], up to 70 Xilinx Virtex-4 FX60 FPGAs are accommodated within one ATCA shelf. Therefore many TPU and RRU cores can be instantiated and distributed on the FPGA cluster. They will work together to cope with the enormous raw data rate from the particle detectors, in a fashion of Single-Instruction-Multiple-Data (SIMD). One ATCA shelf full of 70 FPGAs implies an equivalent processing capability of thousands of commodity PCs for the particle recognition computation in the HADES experiment.

Here the performance data are rough estimation based on our early-stage measurements. Actual performance verification can only be conducted and will be scheduled after the entire system construction is completed.

References

[1] S. Wong, S. Vassiliadis and S. D. Cotofana, "Future Directions of (Programmable and Reconfigurable) Embedded Processors", *In Proc. of the International Samos Workshop on Systems, Architectures, Modeling, and Simulation*, Jul. 2002.

[2] P. Dillien, "An Overview of FPGA Market Dynamics", SOCcentral webpage, 2009.

[3] G. Estrin, Organization of Computer Systems – The Fixed Plus Variable Structure Computer, *In Proc. of the Western Joint Computer Conference*, New York, 1960, pp. 33-40.

[4] G. Estrin, Reconfigurable Computer Origins: The UCLA Fixed-Plus-Variable (F+V) Structure Computer, *IEEE Annals of the History of Computing*, vol. 24, no. 4, Oct.-Dec. 2002, pp. 3-9.

[5] N. Bhatia, S. R. Alam, and J. S. Vetter, "Performance Modeling of Emerging HPC Architectures", *In Proc. of HPCMP Users Group Conference*, pp. 367-373, 2006.

[6] J. Fernando, D. Dalessandro, A. Devulapalli, and K. Wohlever, "Accelerated FPGA Based Encryption", *In Proc. of the Cray Users Group Conference*, 2005.

[7] C. B. Cameron, "Using FPGAs to Supplement Ray-Tracing Computations on the Cray XD-1", *In Proc. of the DoD High Performance Computing Modernization Program Users Group Conference*, pp. 359-363, 2007.

[8] C. Hinkelbein, A. Kugel, R. Manner, M. Muller, M. Sessler, H. Simmler, and H. Singpiel, "Pattern Recognition Algorithms on FPGAs and CPUs for the ATLAS LVL2 Trigger", *IEEE Transactions on Nuclear Science*, vol. 48, no. 3, Part 1, pp. 296-301, 2001.

[9] Dinigroup, "www.dinigroup.com".

[10] C. Chang, J. Wawrzynek, and R. W. Brodersen, "BEE2: a high-end reconfigurable computing system", *IEEE Design and Test of Computers*, vol. 22, no. 2, pp. 114-125, 2005.

[11] A. Tkachenko, D. Cabric, and R. W. Brodersen, "Cognitive Radio Experiments using Reconfigurable BEE2", *In Proc. of the Fortieth Asilomar Conference on Signals, Systems, and Computers*, pp. 2041-2045, 2006.

[12] BEEcube, "www.beecube.com".

[13] T. Gueneysu, T. Kasper, M. Novotny, and C. Paar, "Cryptanalysis with COPACOBANA", *Transactions on Computers*, pp. 1498-1513, 2008.

[14] O. Mencer, K. H. Tsoi, S. Craimer, T. Todman, W. Luk, and M. Y. Wong, "CUBE: A 512-FPGA cluster", *In Proc. of the Southern Programmable Logic Conference*, pp. 51-57, 2009.

[15] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs", *In Proc. of the International Symposium on Field-Programmable Gate Arrays*, Feb. 2006.

[16] S. Lu, P. Yiannacouras, T. Suh, R. Kassa, and M. Konow, "A Desktop Computer with a Reconfigurable *Pentium®*", *ACM Transactions on Reconfigurable Technology and Systems*, vol. 1, no. 1, Mar. 2008.

[17] P. S. Zuchowski, et al., "A Hybrid ASIC and FPGA Architecture", *In Proc. of the International Conference on Computer-Aided Design*, Nov. 2002.

[18] S. J. Wilton, et al., "Design Considerations for Soft Embedded Programmable Logic Cores", *IEEE Journal of Solid-State Circuits*, vol. 40, no. 2, Feb. 2005.

[19] F. Sironi, M. Triverio, H. Hoffmann, M. Maggio, and M. D. Santambrogio, "Self-aware Adaptation in FPGA-based Systems", *In Proc. of the International Conference on Field Programmable Logic and Applications*, Sep. 2010.

[20] A. Agarwal, "Self-aware Computing", Massachusetts Institute of Technology, Final Technical Report, AFRL-RI-RS-TR-2009-161, Jun. 2009.

[21] C. Kao, "Benefits of Partial Reconfiguration", *Xcell Journal*, Fourth Quarter 2005, pp. 65-67.

[22] E. J. Mcdonald, "Runtime FPGA Partial Reconfiguration", *In Proc. of 2008 IEEE Aerospace Conference*, pp. 1-7, Mar. 2008.

[23] C. Choi and H. Lee, "An Reconfigurable FIR Filter Design on a Partial Reconfiguration Platform", *In Proc. of the First International Conference on Communications and Electronics*, pp. 352-355, Oct. 2006.

[24] **M. Liu**, Z. Lu, W. Kuehn, and A. Jantsch, "A Hardware/Software Design Framework for FPGA-based Self-aware Adaptive Computing", under submission.

[25] **M. Liu**, Z. Lu, W. Kuehn, and A. Jantsch, "A Light-weight Routerless Network-on-Chip Infrastructure using FPGA Dynamic Reconfigurability", under submission.

[26] **M. Liu**, Z. Lu, W. Kuehn, and A. Jantsch, "Adaptively Reconfigurable Controller for the Flash Memory", Book of *Flash Memory*, invited book chapter, InTech, ISBN: 978-953-307-272-2, 2011.

[27] **M. Liu**, Z. Lu, W. Kuehn, and A. Jantsch, "Inter-Process Communications using Pipes in FPGA-based Adaptive Computing", *In Proc. of the IEEE Computer Society Annual Symposium on VLSI*, pp. 80-85, Jul. 2010.

[28] **M. Liu**, Z. Lu, W. Kuehn, and A. Jantsch, "Reducing FPGA Reconfiguration Time Overhead using Virtual Configurations", *In Proc. of the International Workshop on Reconfigurable Communication Centric System-on-Chips*, May 2010.

[29] **M. Liu**, Z. Lu, W. Kuehn, and A. Jantsch, "FPGA-based Adaptive Computing Architecture for Correlated Multi-stream Processing", *In Proc. of the Design, Automation and Test in Europe conference*, pp. 973-976, Mar. 2010.

[30] **M. Liu**, Z. Lu, W. Kuehn, S. Yang and A. Jantsch, "A Reconfigurable Design Framework for FPGA Adaptive Computing", *In Proc. of the International Conference on ReConFigurable Computing and FPGAs*, pp. 439-444, Dec. 2009.

[31] **M. Liu**, W. Kuehn, Z. Lu, and A. Jantsch, "Run-time Partial Reconfiguration Speed Investigation and Architectural Design Space Exploration",

In Proc. of the International Conference on Field Programmable Logic and Applications, pp. 498-502, Aug. 2009.

[32] **M. Liu**, W. Kuehn, S. Lange, S. Yang, J. Roskoss, Z. Lu, A. Jantsch, Q. Wang, H. Xu, D. Jin, and Z. Liu, "A High-end Reconfigurable Computation Platform for Nuclear and Particle Physics Experiments", *Computing in Science and Engineering*, vol. 13, no. 2, pp. 52-63, Mar./Apr. 2011.

[33] Q. Wang, A. Jantsch, D. Jin, A. Kopp, W. Kuehn, J. Lang, S. Lange, L. Li, **M. Liu**, Z. Liu, Z. Lu, D. Muechow, J. Roskoss, and H. Xu, "Hardware/Software Co-design of an ATCA-based Computation Platform for Data Acquisition and Triggering", *In Proc. of the IEEE NPSS Real Time Conference*, pp. 485-489, May 2009.

[34] **M. Liu**, J. Lang, S. Yang, T. Perez, W. Kuehn, H. Xu, D. Jin, Q. Wang, L. Li, Z. Liu, Z. Lu, and A. Jantsch, "ATCA-based Computation Platform for Data Acquisition and Triggering in Particle Physics Experiments", *In Proc. of the International Conference on Field Programmable Logic and Applications 2008*, pp. 287-292, Sep. 2008.

[35] **M. Liu**, W. Kuehn, Z. Lu, A. Jantsch, S. Yang, T. Perez, and Z. Liu, "Hardware/Software Co-design of a General-Purpose Computation Platform in Particle Physics", *In Proc. of the IEEE International Conference on Field Programmable Technology*, pp. 177-183, Dec. 2007.

[36] W. Kuehn, C. Gilardi, D. Kirschner, J. Lang, S. Lange, **M. Liu**, T. Perez, L. Schmitt, D. Jin, L. Li, Z. Liu, Y. Lu, Q. Wang, S. Wei, H. Xu, D. Zhao, K. Korcyl, J. T. Otwinowski, P. Salabura, I. Konorov, and A. Mann, "FPGA-Based Compute Nodes for the PANDA Experiment at FAIR", *In Proc. of the NPSS Real Time Conference*, pp. 1-2, Apr. 2007.

[37] T. Perez, C. Gilardi, **M. Liu**, and S. Yang, "A FPGA-based Compute Node for the PANDA Data Acquisition and Trigger System", *In Proc. of the International Winter Meeting on Nuclear Physics*, Apr. 2007.

[38] **M. Liu**, Z. Lu, W. Kuehn, and A. Jantsch, "FPGA-based Particle Recognition in the HADES Experiment", *IEEE Design and Test of Computers*, special issue on Design Methods and Tools for FPGA-Based Acceleration of Scientific Computing, Jul./Aug. 2011 (accepted).

[39] **M. Liu**, Z. Lu, W. Kuehn, and A. Jantsch, "FPGA-based Cherenkov Ring Recognition in Nuclear and Particle Physics Experiments", *In Proc.*

of the International Symposium on Applied Reconfigurable Computing, pp. 169-180, Mar. 2011.

[40] **M. Liu**, A. Jantsch, D. Jin, A. Kopp, W. Kuehn, J. Lang, L. Li, S. Lange, Z. Liu, Z. Lu, D. Muenchow, V. Penchenov, J. Roskoss, S. Spataro, Q. Wang, and H. Xu, "Trigger Algorithm Development on FPGA-based Compute Node", *In Proc. of the IEEE NPSS Real Time Conference*, pp. 478-484, May 2009.

[41] **M. Liu**, W. Kuehn, Z. Lu, and A. Jantsch, "System-on-an-FPGA Design for Real-time Particle Track Recognition and Reconstruction in Physics Experiments", *In Proc. of the EUROMICRO Conference on Digital System Design*, pp. 599-605, Sep. 2008.

[42] D. G. Kirschner, G. Agakishiev, **M. Liu**, T. Perez, W. Kuehn, V. Pechenov, and S. Spataro, "Level 3 Trigger Algorithm and Hardware Platform for the HADES Experiment", *Nuclear Instruments and Methods in Physics Research A*, vol. 598, no. 2, pp. 598-604, 2008.

[43] Z. Lu, **M. Liu**, and A. Jantsch, "Layered Switching for Networks on Chip", *In Proc. of the Design Automation Conference*, pp. 122-127, Jun. 2007.

[44] High Acceptance Di-Electron Spectrometer (HADES) @ GSI 2008. Darmstadt, Germany, "www-hades.gsi.de".

[45] antiProton ANnihilations at DArmstadt (PANDA) @ GSI 2008. Darmstadt, Germany, "www.gsi.de/panda".

[46] BEijing Spectrometer (BES) @ IHEP 2008. Beijing, China, "http://bes.ihep.ac.cn/bes3/index.html".

[47] The Large Hadron Collider (LHC) @ CERN 2008. Geneva, Switzerland, "http://lhc.web.cern.ch/lhc/".

[48] Van der Wolf, P. 2007 " Applications and Memory Organization", *Design Automation and Test Conference (DATE) Tutorial-NoCs at the Age of six.*

[49] I. Froehlich, A. Gabriel, D. Kirschner, J. Lehnert, E. Lins, M. Petri, T. Perez, J. Ritman, D. Schaefer, A. Toia, M. Traxler, and W. Kuehn, "Pattern recognition in the HADES spectrometer: an application of FPGA technology in nuclear and particle physics", *In Proc. of the 2002 IEEE International Conference on Field-Programmable Technology*, pp. 443-444, Dec. 2004.

[50] M. Traxler, "Real-time dilepton selection for the HADES spectrometer", November 2001, Ph.D thesis, II. Physikalisches Institute of Justus-Liebig-Universität Giessen.

[51] C. Hinkelbein, A. Kugel, R. Manner, M. Muller, M. Sessler, H. Simmler and H. Singpiel, "Pattern recognition algorithms on FPGAs and CPUs for the ATLAS LVL2 trigger", *IEEE Transactions on Nuclear Science*, vol. 48, no. 3, Part 1, pp. 296-301, Jun. 2001.

[52] R. Merl, F. Gallegos, C. Pillai, F. Shelley, B. J. Sanchez and A. Steck, "High speed EPICS data acquisition and processing on one VME board", *In Proc. of the 2003 Particle Accelerator Conference*, vol. 4, pp. 2518-2520, May. 2003.

[53] Y. Tsujita, J. S. Lange, and C. Fukunaga, "Construction of a compact DAQ-system using DSP-based VME modules", *In Proc. of the 11th IEEE NPSS Real Time Conference*, pp. 95-98, Jun. 1999.

[54] M. Drochner, W. Erven, P. Wustner, and K. Zwoll, "The second generation of DAQ-Systems at COSY", *IEEE Transactions on Nuclear Science*, vol. 45, no. 4, Part 1, pp. 1882-1888, Aug. 1998.

[55] Y. Nagasaka, I. Arai and K. Yagi, "Data acquisition and event filtering by using transputers", *In Proc. of the Nuclear Science Symposium and Medical Imaging Conference 1991*, pp. 841-844, Nov. 1991.

[56] S. Anvar, F. Bugeon, P. Debu, J.L. Fallou, H. Le Provost, F. Louis, M. Mur, S. Schanne, G. Tarte and B. Vallage, "The Charged Trigger System of NA48 at CERN", *Nuclear Instruments and Methods in Physics Research A*, vol. 419, no. 2-3, pp. 686-694, 1998.

[57] A. Gregerson, "FPGA Design Analysis of the Clustering Algorithm for the CERN Large Hadron Collider", *In Proc. of the IEEE Symposium on Field Programmable Custom Computing Machines*, Apr. 2009.

[58] PCI Industrial Computers Manufactures Group (PICMG), "PICMG 3.0 Advanced Telecommunications Computing Architecture (ATCA) specification", Dec. 2002.

[59] T. Ito, K. Mishou, Y. Okuyama, and K. Kuroda, "A Hardware Resource Management System for Adaptive Computing on Dynamically Reconfigurable Devices", *In Proc. of the Japan-China Joint Workshop on Frontier of Computer Science and Technology*, pp. 196-202, Nov. 2006.

[60] F. Dittmann and M. Goetz, "Applying Single Processor Algorithms to Schedule Tasks on Reconfigurable Devices Respecting Reconfiguration Times", *In Proc. of the 20th International Parallel and Distributed Processing Symposium*, Apr. 2006.

[61] H. Walder and M. Platzner, "Online Scheduling for Block-partitioned Reconfigurable Devices", *In Proc. of the Design Automation and Test in Europe Conference and Exhibition*, pp. 290-295, Dec. 2003.

[62] H. Walder and M. Platzner, "Non-preemptive Multitasking on FPGAs: Task Placement and Footprint Transform", *In Proc. of the 2nd International Conference on Engineering of Reconfigurable systems and Architectures*, pp. 24-30, Jun. 2002.

[63] C. Steiger, H. Walder, and M. Platzner, "Operating Systems for Reconfigurable Embedded Platforms: Online Scheduling of Real-Time Tasks", *IEEE Transactions on Computers*, pp. 1393-1407, Nov. 2004.

[64] H. K. So, A. Tkachenko, and R. Brodersen, "A Unified Hardware/Software Runtime Environment for FPGA-based Reconfigurable Computers using BORPH", *In Proc. of the 4th International Conference on Hardware/Software Codesign and System Synthesis*, pp. 259-264, Oct. 2006.

[65] F. Duhem, F. Muller and P. Lorenzini, "FaRM: Fast Reconfiguration Manager for Reducing Reconfiguration Time Overhead on FPGA", *In Proc. of the International Symposium on Applied Reconfigurable Computing*, pp. 253-260, Mar. 2011.

[66] J. Delorme, A. Nafkha, P. Leray and C. Moy, "New OPBHWICAP Interface for Realtime Partial Reconfiguration of FPGA", *In Proc. of the International Conference on Reconfigurable Computing and FPGAs*, Dec. 2009.

[67] S. Liu, R. N. Pittman, and A. Forin, "Minimizing Partial Reconfiguration Overhead with Fully Streaming DMA Engines and Intelligent ICAP Controller", Technical Report, MSR-TR-2009-150, Microsoft Research, 2009.

[68] H. Kalte and M. Porrmann, "Context Saving and Restoring for Multitasking in Reconfigurable Systems", *In Proc. of the International Conference on Field Programmable Logic and Applications*, Aug. 2005.

[69] C. Huang and P. Hsiung, "Software-controlled Dynamically Swappable Hardware Design in Partially Reconfigurable Systems", *EURASIP Journal on Embedded Systems*, Jan. 2008.

[70] M. Majer, J. Teich, A. Ahmadinia, and C. Bobda, "The Erlangen Slot Machine: A Dynamically Reconfigurable FPGA-Based Computer", *The Journal of VLSI Signal Processing*, vol. 47, no. 1, pp. 15-31, 2007.

[71] S. Fekete, J. van der Veen, M. Majer, J. Teich, "Minimizing Communication Cost for Reconfigurable Slot Modules", *In Proc. of the International Conference on Field Programmable Logic and Applications*, Aug. 2006.

[72] Xilinx Inc., "Early Access Partial Reconfiguration User Guide for ISE 8.1.01i", UG208 (v1.1), Mar. 2006.

[73] Xilinx Inc., "Partial Reconfiguration User Guide", UG702, May, 2010.

[74] Xilinx Inc., "OPB HWICAP (v1.00.b) Product Specification", DS280, Jul. 2006.

[75] Xilinx Inc., "XPS HWICAP (v1.00.a) Product Specification", DS586, Oct. 2007.

[76] Xilinx Inc., "ML405 Evaluation Platform User Guide", UG210, Ma4, 2008.

[77] J. H. Pan, T. Mitra, and W. Wong, "Configuration Bitstream Compression for Dynamically Reconfigurable FPGAs", *In Proc. of the International Conference on Computer-Aided Design*, Nov. 2004.

[78] Y. Birk and E. Fiksman, "Dynamic Reconfiguration Architectures for Multi-context FPGAs", *International Journal of Computers and Electrical Engineering*, vol. 35, no. 6, Nov. 2009.

[79] M. Hariyama, S. Ishihara, N. Idobata and M. Kameyama, "Non-volatile Multi-Context FPGAs using Hybrid Multiple- Valued/Binary Context Switching Signals", *In Proc. of International Conference Reconfigurable systems and Algorithms*, Aug. 2008.

[80] K. Nambaand H. Ito, "Proposal of Testable Multi-Context FPGA Architecture", *IEICE Transactions on Information and Systems*, vol. E89-D, no. 5, May. 2006.

[81] J. Corbet, A. Rubini, and G. Kroah-Hartman, "Linux Device Drivers (Third Edition)", O'REILLY & Associates, Inc., ISBN: 0-596-00590-3.

[82] D. P. Bovet and M. Cesati, "Understanding the Linux Kernel, 3rd Edition", O'REILLY & Associates, Inc., ISBN: 0-596-00565-2, Nov. 2005.

[83] M. Stonebraker, U. Cetintemel, and S. Zdonik, "The 8 Requirements of Real-time Stream Processing", *ACM SIGMOD Record*, vol. 34, no. 4, Dec. 2005.

[84] J. H. Ahn, W. J. Dally, B. Khailany, U. J. Kapasi, and A. Das, "Evaluating the Imagine Stream Architecture", *In Proc. of the Annual International Symposium on Computer Architecture*, Jun. 2004.

[85] U. J. Kapasi, S. Rixner, W. J. Dally, B. Khailany, J. H. Ahn, P. Mattson, and J. D. Owens, "Programmable Stream Processors", *IEEE Computer*, pp. 54-62, Aug. 2003.

[86] U. J. Kapasi, W. J. Dally, S. Rixner, J. D. Owens and B. Khailany, "The Imagine Stream Processor", *IEEE International Conference on Computer Design*, Sep. 2002.

[87] J. Zhu, I. Sander and A. Jantsch, "Performance Analysis of Reconfiguration in Adaptive Real-time Streaming Applications", *In Proc. of the 6th Workshop on Embedded Systems for Real-time Multimedia*, Oct. 2008.

[88] J. Zhu, I. Sander and A. Jantsch, "Buffer Minimization of Real-Time Streaming Applications Scheduling on Hybrid CPU/FPGA Architectures", *In Proc. of Design Automation and Test in Europe*, 2009.

[89] J. Zhu, I. Sander and A. Jantsch, "Constrained Global Scheduling of Streaming Applications on MPSoCs", *In Proc. of the conference on Asia South Pacific Design Automation*, 2010.

[90] E. A. Lee and D. G. Messerschmitt, "Synchronous Data Flow", *In Proc. of the IEEE*, vol. 75, no. 9, Sep. 1987.

[91] E. A. Lee and D. G. Messerschmitt, "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing", *IEEE Transactions on Computers*, vol. C-36, no. 1, Jan. 1987.

[92] P. A. Cherenkov, "Visible Radiation Produced by Electrons Moving in a Medium with Velocities Exceeding that of Light", *Physics Review*, vol. 52, pp. 378-379, Aug. 1937.

[93] I. Cidon and I. Keidar, "Zooming in on Network-on-Chip Architectures", *Lecture Notes in Computer Science*, ISBN: 978-3-642-11475-5, vol. 5869/2010, Jan. 2010.

[94] Arteris Inc., "A comparison of Network-on-Chip and Buses", White paper, 2005.

[95] A. Agarwal, C. Iskander, and R. Shankar, "Survey of Network on Chip (NoC) Architectures & Contributions", *Journal of Engineering, Computing and Architecture*, vol. 3, no. 1, 2009.

[96] A. Kumar, A. Hansson, J. Huisken, and H. Corporaal, "An FPGA Design Flow for Reconfigurable Network-Based Multi-Processor Systems on Chip", *In Proc. of the Design, Automation & Test in Europe Conference & Exhibition*, Apr. 2007.

[97] A. Ehliar, J. Eilert, and D. Liu, "A Comparison of Three FPGA Optimized NoC Architectures", *In Proc. of the Swedish System-on-Chip Conference*, May 2007.

[98] B. Sethuraman, P. Bhattacharya, J. Khan, and R. Vemuri, "LiPaR: A Light-weight Parallel Router for FPGA-based Networks-on-Chip", *In Proc. of the Great Lakes Symposium on VLSI*, Apr. 2005.

[99] A. Kumar, I. Ovadia, J. Huiskens, H. Corporaal, J. van Meerbergen, and Y. Ha, "Reconfigurable Multi-Processor Network-on-Chip on FPGA", *In Proc. of the Annual Conference of the Advanced School for Computing and Imaging*, 2006.

[100] A. S. Lee and N. W. Bergmann, "On-chip Communication Architectures for Reconfigurable System-on-Chip", *In Proc. of the International Conference on Field-Programmable Technology*, Dec. 2003.

[101] IBM Corporation, "The CoreConnect Bus Architecture", www.chips.ibm.com, 1999.

[102] ARM Corporation, "AMBA specification", www.arm.com, 1999.

[103] T. Seceleanu, "Communication on a Segmented Bus Platform", *In Proc. of the IEEE international SOC conference*, Apr. 2004.

[104] W. B. Jone, J. S. Wang, H. I. Lu, I. P. Hsu, and J. Y. Chen, "Design Theory and Implementation for Low-Power Segmented Bus Systems", *ACM Transactions on Design Automation of Electronic Systems*, vol. 8, no. 1, Jan. 2003.

[105] F. Arifin, "Implementation and Evaluation of Segmented-Bus Architecture", *Master Thesis*, IMIT/LECS-2004-72, Dec. 2004.

[106] K. Sekar, K. Lahiri, A. Raghunathan, and S. Dey, "FLEXBUS: A High-Performance System-on-Chip Communication Architecture with a Dynamically Configurable Topology", *In Proc. of the Design Automation Conference*, Jun. 2005.

[107] L. Moeller, I. Grehs, N. Calazans, and F. Moraes, "Reconfigurable Systems Enabled by a Network-on-Chip", *In Proc. of the International Conference on Field Programmable Logic and Applications*, Aug. 2006.

[108] M. B. Stensgaard and J. Sparso, "ReNoC: A Network-on-Chip Architecture with Reconfigurable Topology", *In Proc. of the Second ACM/IEEE International Symposium on Network-on-Chip*, Apr. 2008.

[109] V. Rana, D. Atienza, M. D. Santambrogio, D. Sciuto, and G. D. Micheli, "A Reconfigurable Network-on-Chip Architecture for Optimal Multi-Processor SoC Communication", *In Proc. of the 16th IFIP/IEEE International Conference on VLSI and System-on-Chip*, Oct. 2008.

[110] B. Ahmad, A. T. Erdogan, and S. Khawam, "Architecture of a Dynamically Reconfigurable NoC for Adaptive Reconfigurable MPSoC", *In Proc. of the First NASA/ESA Conference on Adaptive Hardware and Systems*, Jun. 2006.

[111] J. Shen, C. Huang, and P. Hsiung, "Learning-based Adaptation to Applications and Environments in a Reconfigurable Network-on-Chip", *In Proc. of the Design, Automation & Test in Europe Conference & Exhibition*, Mar. 2010.

[112] M. Modarressi, A. Tavakkol, and H. Sarbazi-Azad, "Virtual Point-to-Point Connections for NoCs", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 6, Jun. 2010.

[113] A. Hemani and M. A. Shami, "Move Logic Not Data : A Conceptual Presentation", *In Proc. of the First International Workshop on Network on Chip Architectures*, Nov. 2008.

[114] M. Liu, "Improving the Performance of a Wormhole Router and Wormhole Flow Control", Master Thesis, IMIT/LECS-2005-80, Royal Institute of Technology, Sweden, Dec. 2005.

[115] A. A. Chien, "A Cost and Speed Model for K-ary N-cube Wormhole Routers", *In Proc. of Hot Interconnects*, Aug. 1993.

[116] L. S. Peh and W. J. Dally, "A Delay Model for Router Microarchitectures", *IEEE Micro*, vol. 21, no. 1, Jan./Feb. 2001.

[117] K. Goossens, J. Dielissen, and A. Radulescu, "Æthereal Network on Chip: Concepts, Architectures, and Implementations", *IEEE Design and Test of Computers*, vol.: 22, no. 5, Sep./Oct. 2006.

[118] K. Paulsson, M. Huebner, S. Bayar, and J. Becker, "Exploitation of Runtime Partial Reconfiguration for Dynamic Power Management in Xilinx Spartan III-based Systems", *In Proc. of the International Workshop on Reconfigurable Communication-centric System-on-Chip*, Jun. 2007.

[119] N. Manjikian and E. Cote, "Implementation of a Configurable Crossbar Switch for Prototyping of Single-Chip Multiprocessors", *In Proc. of the IEEE North-East Workshop on Circuits and Systems*, Jun. 2006.

[120] B. Hong, K. Cho, S. Kang, S. Lee, and J. Cho, "On the Configurable Multiprocessor SoC Platform with Crossbar Switch", *In Proc. of the IEEE Asia Pacific Conference on Circuits and Systems*, Dec. 2006.

[121] W. J. Dally and B. Towles, "Principles and Practices of Interconnection Network", Morgan Kaufmann, ISBN 13: 978-0-12-200751-4, Dec. 2003.

[122] N. Chin-Ee and N. Soin, "A Study on Circuit Switching Merits in the Design of Network-on-Chip", *In Proc. of the International Conference on Computer and Communication Engineering*, May. 2010.

[123] M. Lis, K. S. Shim, M. H. Cho, and S. Devadas, "Guaranteed In-order Packet Delivery using Exclusive Dynamic Virtual Channel Allocation", *MIT CSAIL Technical Report*, MIT-CSAIL-TR-2009-036, 2009.

[124] R. Mullins, A. West, and S. Moore, "The Design and Implementation of a Low-Latency On-Chip Network", *In Proc. of the Asia and South Pacific Conference on Design Automation*, Jan. 2006.

[125] Xilinx Inc., "Virtex-4 FPGA Configuration User Guide", UG071, Apr. 2008.

[126] S. Liu, R. N. Pittman, and A. Forin, "Energy Reduction with Run-Time Partial Reconfiguration", *Technical Report of Microsoft Research*, MSR-TR-2009- 2017, Sep. 2009.

[127] R. Shashikumar and L. C. S. Gouda, "Self-healing Reconfigurable FPGA Based Fault Tolerant Security Model for Shared Internet Resources", *In Proc. of the International Journal of Computer Science and Network Security*, vol. 9, no. 1, Jan. 2009.

[128] A. Akoglu, A. Sreeramareddy, and J. G. Josiah, "FPGA based Distributed Self healing Architecture for Reusable Systems", *Cluster Computing*, vol. 12, no. 3, pp. 269-284, 2009.

[129] H. Psaier and S. Dustdar, "A Survey on Self-healing Systems: Approaches and Systems", *Computing*, ISSN (Print) 1436-5057, ISSN (Online) 0010-485X, Nov. 2010.

[130] D. Koch, C. Beckhoff, and J. Teich, "Minimizing Internal Fragmentation by Fine-grained Two-dimensional Module Placement for Runtime Reconfigurable Systems", *In Proc. of the IEEE Symposium on Field Programmable Custom Computing Machines*, 2009.

[131] D. Koch, C. Beckhoff, and J. Torrison, "Fine-grained Partial Runtime Reconfiguration on Virtex-5 FPGAs", *In Proc. of the IEEE Symposium on Field Programmable Custom Computing Machines*, 2010.

[132] A. X. Widmer, P. A. Franaszek, "A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code", *IBM Journal of Research and Development*, vol. 27, no. 5, 1983, pp. 440-451.

[133] Xilinx, Inc., Multi-Port Memory Controller (MPMC) (v4.01.a), DS643, March 12, 2008.

[134] J. Lehnert, et al., Ring Recognition in the HADES Second-level Trigger, *Nuclear Instruments and Methods in Physics Research A 433*, pp. 268-273, 1999.

[135] J. Lehnert, et al., Performance of the HADES Ring Recognition Hardware, *Nuclear Instruments and Methods in Physics Research A 502*, pp. 261-265, 2003.