

Advanced Multicore Systems-On-Chip

Abderazek Ben Abdallah

Advanced Multicore Systems-On-Chip

Architecture, On-Chip Network, Design

 Springer

Abderazek Ben Abdallah
School of Computer Science
and Engineering
The University of Aizu
Aizu-Wakamatsu, Fukushima
Japan

ISBN 978-981-10-6091-5 ISBN 978-981-10-6092-2 (eBook)
DOI 10.1007/978-981-10-6092-2

Library of Congress Control Number: 2017948616

© Springer Nature Singapore Pte Ltd. 2017

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

This Springer imprint is published by Springer Nature

The registered company is Springer Nature Singapore Pte Ltd.

The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

To Sonia, Tesnim and Beyram.

Preface

Nowadays, the technology has become an essential pawn in our life that is not restricted anymore to academic research or critical missions; but it is moving away to provide the simplest and easiest services that we need or desire for our daily life. With the expanse of technology and the rising of new trends every day, the necessity to process information anywhere and anytime is becoming the main goal of developers and manufacturers.

Systems on chip (SoCs) are embedded systems composed of several modules (processors, memories, input/output peripherals, etc.) on a single chip. With SoCs, it is now possible to process information and execute critical tasks at higher speed and lower power on a tiny chip. This is due to the increasing number of transistors that can be embedded on a single chip, which keeps doubling approximately every 2 years as Intel co-founder Gordon Moore predicted in 1965. This made shrinking the chip size while maintaining high performance possible. This technology scaling has allowed SoCs to grow continuously in component count and complexity and evolve to systems with many processors embedded on a single SoC. With such high integration level available, the development of multi and many cores on a single die has become possible.

Historically, the SoCs paradigm has evolved from fairly simple uncore single memory designs to complex homogeneous/heterogeneous multicore SoC (MCSoc) systems consisting of a large number of intellectual property (IP) cores on the same silicon. To meet the challenges arising from high computational demands posed by latest state-of-the-art embedded and consumer electronic devices, most current systems are based on such paradigm, which represents a real revolution in many aspects of computing.

The attraction of multicore processing for power reduction is compelling in embedded and in general purpose computing. By splitting a set of tasks among multiple cores, the operating frequency necessary for each core can be reduced, thereby facilitating a reduction in the voltage on each core. As dynamic power is proportional to the frequency and to the square of the voltage, we are able to obtain a sizable gain, even though we may have more cores running.

As more and more cores are integrated into these designs to share the ever increasing processing load, the primary challenges are geared toward efficient memory hierarchy, scalable system interconnect, new programming models, and efficient integration methodology for connecting such heterogeneous cores into a single system capable of leveraging their individual flexibility.

Current design methods are inclined toward mixed hardware/software (HW/SW) co-designs, targeting multicore SoCs for application specific domains. To decide on the lowest cost mix of cores, designers must iteratively map the device's functionality to a particular HW/SW partition and target architectures. In addition, to connect the heterogeneous cores, the architecture requires high performance-based complex communication architectures and efficient communication protocols, such as hierarchical bus, point-to-point connection, or the recent new interconnection paradigm—network on chip.

Software development also becomes far more complex due to the difficulties in breaking a single processing task into multiple parts that could be processed separately and then reassembled later. This reflects the fact that certain processor jobs could not possibly be easily parallelized to run concurrently on multiple processing cores and that load balancing between processing cores—especially heterogeneous cores—is extremely difficult.

This book is organized into nine chapters. The book stands independent and we have made every attempt to make each chapter self-contained as well.

Chapter 1 introduces multicore systems on chip (MCSocS) architectures and explores SoCs technology and the challenges it presents to organizations and developers building next-generation multicore SoCs-based systems.

Understanding the technological landscape and design methods in some level of details are very important. This is because so many design decisions in multicore architecture today are guided by the impact of the technology. Chapter 2 presents design challenges and conventional design methods of MCSocS. It also describes a so-called scalable core-based method for systematic design environment of application specific heterogeneous multicore SoC architectures. The architecture design used in conventional methods of multicore SoCs and custom multiprocessor architectures are not flexible enough to meet the requirements of different application domains and not scalable enough to meet different computation needs and different complexities of various applications. Therefore, designers should be aware of existing design methods and also be ready to innovate or adapt appropriate design methods for individual target platform.

Understanding the software and hardware building blocks and the computation power of individual components in these complex MCSocS is necessary for designing power-, performance-, and cost-efficient systems. Chapter 3 describes in details the architectures and functions of the main building blocks that are used to build such complex multicore SoCs. Readers with a relevant background in multicore SoC building blocks could effectively skip some of the materials mentioned in this chapter. The knowledge of these aspects is not an absolute requirement for understanding the rest of the book, but it does help novice students or beginners to

get a glimpse of the big picture of a heterogeneous or homogeneous MCSoC organization.

Whether homogeneous, heterogeneous, or hybrid multicore SoCs, IP cores must be connected in a high-performance, scalable, and flexible manner. The emerging technology that targets such connections is called an on-chip interconnection network, also known as a network on chip (NoC), and the philosophy behind the emergence of such innovation has been summarized by William Dally at Stanford University as *route packets, not wires*.

Chapters 4–6 presents fundamental and advanced on-chip interconnection network technologies for multi- and many-core SoCs. These three chapters are all very important part of the book since they allow the reader to understand what needed microarchitecture for on-chip routers and network interfaces are essential towards meeting latency, area, and power constraints. Reader will also understand practical issues about what system architecture (topology, routing, flow control, NI, and 3D integration) is most suited for these on-chip networks.

With the rise of multicore and many-core systems, concurrency becomes a major issue in the daily life of a programmer. Thus, compiler and software development tools will be critical towards helping programmers create high-performance software. Programmers should make sure that their parallelized program codes would not cause race condition, memory-access deadlocks, or other faults that may crash their entire systems. Chapter 7 describes a novel parallelizing compiler design for high-performance computing.

Power dissipation continues to be a primary design constraint and concern in single and multicore systems. Increasing power consumption not only results in increasing energy costs, but also results in high die temperatures that affect chip reliability, performance, and packaging cost. Chapter 8 provides a detailed investigation of power reduction techniques for multicore SoC at components and network levels. Energy conservation has been largely considered in the hardware design, in general and also in embedded multicore system's components, such as CPUs, disks, displays, memories, and so on. Significant additional power savings could be also achieved by incorporating low power methods into the design of network protocols used for data communication (audio, video, etc.).

Chapter 9 ties together previous chapters and presents a real embedded multicore SoC system design targeted for elderly health monitoring. For this book, we used our experience to illustrate the complete design flow for a multicore SoC running an electrocardiogram (ECG) application in parallel. Thanks to the recent technological advances in wireless networking, embedded microelectronics, and the Internet, computer and biomedical scientists are now capable to fundamentally modernize and change the way health care services are deployed. Discussions on how to design the algorithms, architecture, register-transfer level implementation, and FPGA prototyping and validation for ECG processing are presented in details.

This book took nearly 2 years to complete. It evolved from our first book and is derived from our teaching experiences in embedded system designs and architecture to both undergraduate and graduate students. Multicore systems paradigm created stupendous opportunities to increase overall system performance, but also created

many design challenges that designers must now overcome. Thus we must continue innovating new algorithms and techniques to solve these challenges.

The author is thankful to numerous colleagues and graduate students for their lively discussions and their help in preparing the manuscript of this book. Special thanks are due to the publishers in bringing out this book quickly, yet maintaining very high quality.

Aizu-Wakamatsu, Japan

Abderazek Ben Abdallah

Contents

1	Introduction to Multicore Systems On-Chip	1
1.1	The Multicore Revolution	1
1.1.1	The Impact of Moore’s Law	2
1.1.2	On-Chip Interconnection Schemes	2
1.1.3	Parallelism and Performance	4
1.1.4	Parallel Hardware Architectures	6
1.1.5	The Need for Multicore Computing	8
1.1.6	Multicore SoCs Potential Applications	8
1.2	Multicore SoC Basics	10
1.2.1	Programmability Support	10
1.2.2	Software Organization	11
1.2.3	Programming Multicore Systems	12
1.2.4	Multicore Implementations	13
1.3	Multicore SoCs Design Challenges	15
1.3.1	Cache Coherence	15
1.3.2	Power and Temperature	16
1.3.3	Multi-threading and Memory Management	16
1.3.4	On-Chip Interconnection Networks	17
1.3.5	Reliability Issues	17
1.4	Chapter Summary	17
	References	18
2	Multicore SoCs Design Methods	19
2.1	Introduction	19
2.2	Design Space Exploration	20
2.3	Parallel Software Development Phase	22
2.3.1	Compiler-Based Schemes	22
2.3.2	Language Extensions Schemes	23
2.3.3	Language Extensions with APIs	24
2.3.4	Model-Based Schemes	24

2.4	Generic Architecture Template (GAT) for Real Multicore SoC	
	Design	25
2.4.1	Target Multicore SoC Platform	25
2.4.2	Design Method	26
2.4.3	QueueCore Architecture	28
2.4.4	Performance Analysis	33
2.5	Chapter Summary	35
	References.	35
3	Multicore SoC Organization	39
3.1	Introduction	39
3.1.1	Heterogeneous MCSoC	41
3.1.2	Homogeneous MCSoC	43
3.1.3	Multicore SoC Applications	44
3.1.4	Applications Mapping	45
3.2	MCSoC Building Blocks	46
3.2.1	Processor Core	48
3.2.2	Memory	48
3.2.3	Cache	50
3.2.4	Communication Protocols	50
3.2.5	Intellectual Property (IP) Cores	52
3.2.6	IP Cores with Multiple Clock Domains	53
3.2.7	Selection of IP Cores	54
3.3	MCSoC Memory Hierarchy	56
3.3.1	Types on On-Chip Memory	56
3.3.2	Scratchpad Memory	58
3.3.3	Off-Chip Memory	58
3.3.4	Memory Power Reduction in SoC Designs	59
3.4	Memory Consistency in Multicore Systems	61
3.4.1	Cache Coherence Problem	61
3.4.2	Cache Coherence Protocols	63
3.5	Chapter Summary	65
	References.	65
4	Multicore SoC On-Chip Interconnection Networks	67
4.1	Introduction	67
4.2	Network-on-Chip Architecture	69
4.2.1	Topology.	69
4.2.2	Switching	71
4.2.3	Flow Control.	74
4.2.4	Routing Algorithms.	76
4.3	Hardware Design of On-Chip Network.	82
4.3.1	Topology Design.	82
4.3.2	Pipeline Design.	84
4.3.3	Crossbar Design	88

4.3.4	Limitations of Regular Mesh Topology.	89
4.3.5	SPL Insertion Algorithm.	90
4.3.6	Network Interface Design	94
4.4	Chapter Summary	105
	References.	105
5	Advanced Multicore SoC Interconnects.	107
5.1	Introduction	107
5.2	Three-Dimensional On-Chip Interconnect.	109
5.2.1	3D-NoC Versus 2D-NoC	109
5.2.2	Routing Algorithms.	111
5.2.3	Topology Design.	113
5.2.4	Switching Policy.	115
5.2.5	3D-NoC Router Architecture Design.	116
5.2.6	Network Interface Architecture	127
5.2.7	3D-NoC Design Evaluation.	130
5.2.8	Conclusion	143
5.3	Photonic On-Chip Interconnect for High-Bandwidth Multicore SoCs.	143
5.3.1	Photonic Communication Building Blocks	144
5.3.2	Design Challenges.	146
5.3.3	Fault Models	148
5.3.4	Fault-Tolerant Photonic Network-on-Chip Architecture.	150
5.3.5	Evaluation	160
5.3.6	Related Literature	166
5.3.7	Chapter Summary	168
	References.	168
6	3D Integration Technology for Multicore Systems On-Chip	175
6.1	3D Integration Technology.	175
6.2	Fault-Tolerant TSV Cluster for 3D Integration.	177
6.2.1	Fault-Tolerance for TSV-Clusters	178
6.3	Adaptive Online TSV Sharing Algorithm.	182
6.3.1	Weight Generation	185
6.3.2	TSV-Clusters Return	185
6.3.3	Weight Adjustment	186
6.3.4	Design Optimization	187
6.4	Evaluation Results	189
6.4.1	Defect-Rate Evaluation	190
6.4.2	Performance Evaluation.	191
6.4.3	Latency Evaluation	192
6.4.4	Throughput Evaluation	193
6.4.5	Router Hardware Complexity	194
6.4.6	Comparison.	195
6.5	Chapter Summary	197
	References.	197

7	Parallelizing Compiler for Single and Multicore Computing	201
7.1	Introduction	201
7.1.1	Instruction Level Parallelism	202
7.1.2	Queue Computation Model	204
7.2	Parallel Queue Compiler	204
7.2.1	Queue Processor Overview	204
7.2.2	Compiling for One-Offset QueueCore Instruction Set	205
7.3	Parallelizing Compiler Framework	208
7.3.1	One-Offset P-Code Generation Phase	209
7.3.2	Offset Calculation Phase	212
7.3.3	Instruction Scheduling Phase	213
7.3.4	Natural Instruction Level Parallelism Extraction: Statement Merging Transformation	214
7.3.5	Assembly Generation Phase	216
7.4	Parallelizing Compiler Development Results	217
7.4.1	Queue Compiler Evaluation	218
7.4.2	Comparison of Generated QueueCore Code with Optimized RISC Code	220
7.5	Chapter Summary	221
	References	222
8	Power Optimization Techniques for Multicore SoCs	225
8.1	Introduction	225
8.2	Power-Aware Technological-Level Design Optimizations	227
8.2.1	Factors Affecting CMOS Power Consumption	227
8.2.2	Reducing Voltage and Frequency	228
8.2.3	Reducing Capacitance	229
8.3	Power-Aware Logic-Level Design Optimizations	230
8.3.1	Clock Gating	230
8.3.2	Logic Encoding	231
8.3.3	Data Guarding	232
8.4	Power-Aware System Level Design Optimizations	232
8.4.1	Hardware System Architecture Power Consumption Optimizations	233
8.4.2	Operating System Power Consumption Optimization	236
8.4.3	Application, Compilation Techniques, and Algorithm	238
8.4.4	Energy Reduction in Network Protocols	238
8.5	Chapter Summary	242
	References	242
9	Real Deign of Embedded Multicore SoC for Health Monitoring	245
9.1	Introduction	245
9.1.1	Electrocardiography and Heart Diseases	246

- 9.2 Application Specific Digital Signal Processing 248
 - 9.2.1 Analog and Digital Signals 250
 - 9.2.2 Signal Processing 250
 - 9.2.3 Analog to Digital Conversion 250
- 9.3 Period-Peak ECG Detection Algorithm. 251
 - 9.3.1 Period Detection 253
 - 9.3.2 Peaks Detection. 254
- 9.4 Multicore SoC Hardware Design 257
 - 9.4.1 Signal Reading 257
 - 9.4.2 Filtering 258
 - 9.4.3 Data Processing. 259
 - 9.4.4 Processor Core 260
- 9.5 Real-Time Monitoring Interface Design 261
 - 9.5.1 Data Capturing 261
 - 9.5.2 Data Display and Analysis 262
- 9.6 System Hardware and Software Design Evaluation 265
 - 9.6.1 Hardware Complexity 265
 - 9.6.2 Performance Evaluation. 266
- 9.7 Chapter Summary 266
- References. 267
- Index 269**

List of Figures

Fig. 1.1	Scaling of transistor counts and operating frequency in ICs. The feature size and design abstraction are also shown in the graph.	3
Fig. 1.2	Different ways for exploiting parallelism over various system organization: a Single core, b Multicore with separate caches, c Multicore with shared cache, d Multicore with simultaneous threading, e Multiprocessor	6
Fig. 1.3	From PCB to MCSoC.	7
Fig. 1.4	Multicore SoC potential applications.	9
Fig. 1.5	Typical multicore SoC architectural view	10
Fig. 1.6	Software layers on <i>top</i> of the hardware.	12
Fig. 1.7	Sample OpenMP code using <i>section</i> and <i>parallel</i> directives: a Functional decomposition, b Data decomposition	13
Fig. 1.8	Heterogeneous Multicore CELL Organization.	14
Fig. 2.1	SoC typical architecture	21
Fig. 2.2	Compiler-based scheme.	22
Fig. 2.3	Parallel for loop with OpenMP	24
Fig. 2.4	Multicore SoC system platform. This is a typical instance of the architecture, where the addition of a new core will not change the principle of the methodology.	26
Fig. 2.5	Linked task design flow graph (DFG). a Hardware related tasks, b Application related tasks	27
Fig. 2.6	Next QH and QT pointers calculation mechanism.	29
Fig. 2.7	QC-2's source 2 address calculation	30
Fig. 2.8	QC-2's FADD hardware	31
Fig. 2.9	QC-2's FMUL hardware	32
Fig. 2.10	Resource usage and timing for 256*33 bit QREG unit for different coding and optimization strategies.	33

Fig. 2.11 Achievable frequency is the instruction throughput for hardware implementations of the QC-2 processor. Simulation speeds have been converted to a nominal frequency rating to facilitate comparison 34

Fig. 3.1 General organization view of a modern typical MCSoC 40

Fig. 3.2 Example of an embedded multicore system for a typical digital still camera 41

Fig. 3.3 Example of MPEG-2 encoder for a heterogeneous MCSoC system 42

Fig. 3.4 Heterogeneous MCSoC organization example 42

Fig. 3.5 Homogeneous MCSoC organization example 43

Fig. 3.6 Example of MCSoC application in wireless communication: Lucent Daytona MCSoC 45

Fig. 3.7 Simplified view of a typical MCSoC architecture with different core and memory types 47

Fig. 3.8 State-of-the-art MCSoC architecture based on network-on-chip paradigm 47

Fig. 3.9 Typical 5 pipeline stages of a RISC processor core 48

Fig. 3.10 Example of MCSoC with single external DRAM memory 49

Fig. 3.11 Cache organization in a single node of a typical MCSoC 50

Fig. 3.12 Evolution of on-chip communication interconnect 51

Fig. 3.13 Open core protocol (OCP) and Network protocol (NP) interfacing 52

Fig. 3.14 NoC operation 52

Fig. 3.15 Intellectual property example 53

Fig. 3.16 Three clock domains MCSoC 54

Fig. 3.17 Example of mapping of an MPEG-2 decoder. **a** Using two cores, **b** Using three cores 55

Fig. 3.18 Simplified view of a MCSoC architecture having different memories 57

Fig. 3.19 Example of four cores communicating via FIFOs 58

Fig. 3.20 MCSoC memory subsystem with SPARM (only interconnection for one node is shown for simplicity) 59

Fig. 3.21 Projection of memory/logic composition of power-constrained SoC chips [11] 60

Fig. 3.22 Direct-mapped cache organization 62

Fig. 3.23 Cache coherence problem example without coherence protocol 63

Fig. 4.1 Typical paradigms: **a** circuit switching, **b** packet switching 68

Fig. 4.2 Typical NoC topologies 69

Fig. 4.3 Example of a 3×3 NoC based on mesh topology. R: router/switch, PE: processing element, NI: network interface 70

Fig. 4.4 Store-and-forward switching 72

Fig. 4.5 Wormhole switching 73

Fig. 4.6 Virtual-cut-through switching 74

Fig. 4.7 ON/OFF flow control 75

Fig. 4.8 Credit-based flow control 76

Fig. 4.9 ACK/NACK flow control 76

Fig. 4.10 Categorization of routing algorithms according to the number of destinations: **a** unicast, **b** multicast 77

Fig. 4.11 Categorization of routing algorithms according to decision locality: **a** distributed, **b** source 78

Fig. 4.12 Categorization of routing algorithms according to adaptivity: **a** deterministic, **b** adaptive 78

Fig. 4.13 Categorization of routing algorithms according to minimality: **a** minimal, **b** non-minimal. 79

Fig. 4.14 Deadlock example in adaptive NoC systems. 80

Fig. 4.15 Virtual-channel-based router architecture. 80

Fig. 4.16 Virtual-output-queue-based router architecture. 81

Fig. 4.17 4×4 mesh topology 82

Fig. 4.18 External connections to one router 83

Fig. 4.19 ONoC router block diagram 84

Fig. 4.20 Matrix arbitration example 86

Fig. 4.21 Stall-go block diagram 87

Fig. 4.22 **a** State machine design, **b** Nearly full signal output 87

Fig. 4.23 Arbiter control signals. 88

Fig. 4.24 Short-path-link (SPL) insertion example 90

Fig. 4.25 SPL insertion algorithm. 91

Fig. 4.26 Extra-port insertion 92

Fig. 4.27 Dimension reversal with 2 SPLs 93

Fig. 4.28 Hotspot with 2 SPL. 93

Fig. 4.29 JPEG encoder with 3 SPL. 94

Fig. 4.30 Nigh-level view of the network interface 95

Fig. 4.31 Distributed routing NI architecture block diagram. 96

Fig. 4.32 Packet format 98

Fig. 4.33 Packet HEADER format 98

Fig. 4.34 BODY and END format 98

Fig. 4.35 HEADER flit format 99

Fig. 4.36 BODY flit format 99

Fig. 4.37 END flit format. 99

Fig. 4.38 Format of packet header after deflitzation 100

Fig. 4.39 Format of BODY/END flits after deflitzation. 100

Fig. 4.40 Internal structure of NI for distributed routing. 101

Fig. 4.41 C2R-buffer 101

Fig. 4.42 Flitizer module architecture. 102

Fig. 4.43 Core-to-router (C2R) controller architecture 103

Fig. 4.44	Router-to-core (R2C) buffer	103
Fig. 4.45	Deflitzerizer module architecture	104
Fig. 4.46	R2C controller module architecture.	104
Fig. 5.1	SoC interconnection types: a Shared bus, b Point-to-Point, c NoC	110
Fig. 5.2	Configuration example of a $4 \times 4 \times 4$ 3D-ONoC based on mesh topology.	114
Fig. 5.3	3D-ONOC flit format	116
Fig. 5.4	3D-ONoC pipeline stages: buffer writing (BW), routing calculation and switch allocation (RC/SA) and crossbar traversal stage (CT)	117
Fig. 5.5	Input-port module architecture.	119
Fig. 5.6	Switch allocator architecture	122
Fig. 5.7	Stall-Go flow control mechanism	123
Fig. 5.8	Stall-Go flow control finite state machine	123
Fig. 5.9	Scheduling matrix priority assignment.	124
Fig. 5.10	Crossbar circuit	126
Fig. 5.11	Network interface architecture: Transmitter side	128
Fig. 5.12	Network interface architecture: Receiver side	130
Fig. 5.13	Task graph of the JPEG encoder.	131
Fig. 5.14	Extended task graph of the JPEG encoder.	132
Fig. 5.15	JPEG encoder mapped on 2×4 2D-ONoC.	132
Fig. 5.16	JPEG encoder mapped on $2 \times 2 \times 2$ 3D-ONoC.	133
Fig. 5.17	Matrix multiplication example: The multiplication of an i $\times k$ matrix A by a $k \times j$ matrix B results in an $i \times j$ matrix R	133
Fig. 5.18	Simple example demonstrating the matrix multiplication calculation	134
Fig. 5.19	3×3 matrix multiplication using a optimistic and b pessimistic mapping approaches	135
Fig. 5.20	Execution time comparison between 3D- and 2D-ONoC	139
Fig. 5.21	Average number of hops comparison for both pessimistic and optimistic mappings on 3×3 network size.	140
Fig. 5.22	Average number of hops comparison for both pessimistic and optimistic mappings on 4×4 network size.	141
Fig. 5.23	Average number of hops comparison for both pessimistic and optimistic mappings on 6×6 network size.	141
Fig. 5.24	Stall average count comparison between 3D- and 2D-ONoC	142
Fig. 5.25	Stall average count comparison between 3D- and 2D-ONoC with different traffic loads	142
Fig. 5.26	Execution time comparison between 3D- and 2D-ONoC with different traffic loads	142
Fig. 5.27	3D-Stacked photonic network-on-chip architecture	145
Fig. 5.28	Photonic link architecture	145
Fig. 5.29	Gateway organization	146

Fig. 5.30 FT-PHENIC system architecture. **a** 3×3 mesh-based system, **b** 5×5 non-blocking photonic switch, **c** Unified tile including PE, NI, and control modules 150

Fig. 5.31 Microring fault-resilient photonic router (MRPR): **a** Non-blocking fault-tolerant photonic switch, **b** Light-weight control router. 152

Fig. 5.32 Example of how a non-redundant MR’s functionality can be mimicked by redundant ones. 153

Fig. 5.33 Microring fault-resilient path configuration: **a** Path setup, **b** Path-blocked, **c** Faulty MR with recovery. GW_0 : Gateway for data, GW_1 : Gateway for acknowledgment signals, PS: photonic switch, MRCT: Microring Configuration Table, MRST: Microring State Table. 00 = Not faulty, Not blocked, 01 = Not faulty, Blocked, 10 = Faulty. 157

Fig. 5.34 Fault-tolerant path-configuration algorithm 158

Fig. 5.35 Latency comparison results under random uniform traffic: **a** Overall Latency, **b** Latency near saturation 163

Fig. 5.36 Latency results of each system as faults are introduced. 164

Fig. 5.37 Bandwidth comparison results under random uniform traffic. 164

Fig. 5.38 Bandwidth comparison results as faults are introduced 164

Fig. 5.39 Total energy and energy efficiency comparison results under random uniform traffic near saturation. 165

Fig. 5.40 Total energy and energy efficiency comparison results under random uniform traffic with 4% of MRs acting faulty. 165

Fig. 5.41 Example of photonic switches. From left to right: PHENIC’s original [9], crossbar, and crux [104] 167

Fig. 6.1 Reducing footprint and wire length in 3D-stack structure 176

Fig. 6.2 3D integration schemes: **a** Wire bonding; **b** Solder balls; **c** Through silicon vias; **d** Wireless stacking 176

Fig. 6.3 TSV fault-tolerance schemes: **a** Redundancy technique; **b** Double TSV; **c** Network TSV 178

Fig. 6.4 High-level view of the system architecture with $3 \times 3 \times 3$ configuration 180

Fig. 6.5 TSV sharing area placement and connectivity between two neighboring routers 180

Fig. 6.6 The TSV fault-tolerance architecture: **a** Router wrapper; **b** Connection between two layers. *Red rectangles* represent TSVs. *S-UP* and *S-DOWN* are the sharing arbitrators which manage the proposed mechanism. *CR* stands for configuration register and *W* is the flit width 181

Fig. 6.7 Adaptive online TSV sharing algorithm 183

Fig. 6.8	An example of the sharing algorithm on a 4×4 layer: a Initial state with ten defected TSV clusters; b Best candidates selection; c Borrowing chain creating and selection refining. d Final result with six disabled routers	184
Fig. 6.9	Example of the weight adjustment performed to disable routers' sharing: a Before weight update; b After weight update.	186
Fig. 6.10	Examples of virtual TSV: a return the TSV cluster to the original router; b borrow a cluster from a higher weight router.	188
Fig. 6.11	Circuit of 1:4 serialization.	189
Fig. 6.12	Defect-rate evaluation: a Layer size: 2×2 (4 routers, 16 TSV clusters); b Layer size: 4×4 (16 routers, 64 TSV clusters); c Layer size: 8×8 (64 routers, 256 TSV clusters); d Layer size: 16×16 (256 routers, 1024 TSV clusters); e Layer size: 32×32 (1024 routers, 4096 TSV clusters); f Layer size: 64×64 (4096 routers, 16384 TSV clusters).	191
Fig. 6.13	Evaluation result: a Average packet latency; b Throughput	193
Fig. 6.14	Single layer layout illustrating the TSV sharing areas (<i>red boxes</i>). The layout size is $865 \mu\text{m} \times 865 \mu\text{m}$	195
Fig. 7.1	Instruction sequence generation from the parse tree of expression $x = \frac{a+b}{b-c}$	205
Fig. 7.2	Instruction sequence generation from DAG of expression $x = \frac{a+b}{b-c}$	207
Fig. 7.3	Parallelizing compiler infrastructure	209
Fig. 7.4	QIR code fragment	214
Fig. 7.5	Statement merging transformation.	215
Fig. 7.6	Assembly output for QueueCore processor.	216
Fig. 7.7	Effect on ILP of statement merging transformation in the queue compiler	218
Fig. 7.8	Instruction level parallelism improvement of queue compiler over optimizing compiler for a RISC machine	220
Fig. 7.9	Normalized code size for two embedded RISC processors and QueueCore	221
Fig. 8.1	Clock gating example	231
Fig. 8.2	Dual operation ALU with guard logic. The multiplexer does the selection only after both units have completed their evaluation. The evaluation of one of the two units is avoided by using a guard logic; two latches (L1 and L2) are placed with enable signals (s1 and s2) at the inputs of the shifter and the adder respectively	232
Fig. 8.3	Power consumption in typical processor core	235
Fig. 8.4	Protocol stack of a generic wireless network, and corresponding areas of energy-efficient possible research	240
Fig. 9.1	A typical ECG wave.	247

Fig. 9.2 Faulty ECG Analysis. 252

Fig. 9.3 PPD algorithm processing flow. 252

Fig. 9.4 Period detection computation details. 254

Fig. 9.5 Peaks detection computation details 254

Fig. 9.6 Period detection: finding maximum value algorithm. The autocorrelation step *ACF_STEP* is set 256 255

Fig. 9.7 Period detection: reduce negative value algorithm. 255

Fig. 9.8 Period detection: find base points 256

Fig. 9.9 Period detection: sort base points 256

Fig. 9.10 High-level view of the BANSMOM system architecture. 257

Fig. 9.11 Prototyped multicore SoC block diagram 260

Fig. 9.12 Nios II core architecture block diagram. 260

Fig. 9.13 Software simulation output 262

Fig. 9.14 (a) Get live-data, (b) Get previous-data. 263

Fig. 9.15 Multicore SoC system running snapshot 264

Fig. 9.16 Interactive RTI tool displaying ECG waves 264

List of Tables

Table 2.1	Linked task description	27
Table 2.2	QC-2 processor design results: modules complexity as LE (logic elements) and TCF (total combinational functions) when synthesized for FPGA (with Stratix device) and Structured ASIC (HardCopy II) families	34
Table 3.1	Cache coherence states	64
Table 4.1	Area utilization for a 5-ports router	91
Table 4.2	Area utilization for 6-port router	92
Table 4.3	Flit Types and Coding	98
Table 4.4	Summary of decisions for distributed routing NI	100
Table 5.1	Simulation parameters	136
Table 5.2	3D-ONoC hardware complexity compared with 2D-ONoC	137
Table 5.3	Microring configuration for normal data transmission	153
Table 5.4	Microring backup configuration for data transmission	154
Table 5.5	Wavelength assignment for acknowledgment signal (Mod: Modulator, and Det: Photodetector)	154
Table 5.6	Various switches and their estimated losses. AL: Average Loss, WL: Worst Loss	155
Table 5.7	Insertion loss parameters for 22 nm process	155
Table 5.8	Configuration parameters	160
Table 5.9	Photonic communication network energy parameters	161
Table 5.10	MR requirement comparison results for 64 cores systems	161
Table 5.11	MRs requirement comparison results for 256-core systems	162
Table 6.1	Configuration register (CR) description	182
Table 6.2	Technology parameters	189
Table 6.3	System configurations	190
Table 6.4	Simulation configurations	192
Table 6.5	Hardware complexity of a single router	194
Table 6.6	Comparison results between the proposed approach and the existing works	196

Table 7.1	Lines of C code for each phase of the queue compiler's back end	218
Table 7.2	Instruction category percentages for the compiled benchmarks for the QueueCore.	219
Table 7.3	QueueCore's program maximum offset reference value.	219
Table 8.1	Operating system functionality and corresponding techniques for optimizing energy utilization	236
Table 9.1	Hardware complexity	265
Table 9.2	Performance evaluation	265