
ATLANTIS AMBIENT AND PERVASIVE INTELLIGENCE

VOLUME 3

SERIES EDITOR: ISMAIL KHALIL

Atlantis Ambient and Pervasive Intelligence

Series Editor:

Ismail Khalil, Linz, Austria

(ISSN: 1875-7669)

Aims and scope of the series

The book series 'Atlantis Ambient and Pervasive Intelligence' publishes high quality titles in the fields of Pervasive Computing, Mixed Reality, Wearable Computing, Location-Aware Computing, Ambient Interfaces, Tangible Interfaces, Smart Environments, Intelligent Interfaces, Software Agents and other related fields. We welcome submission of book proposals from researchers worldwide who aim at sharing their results in this important research area.

All books in this series are co-published with World Scientific.

For more information on this series and our other book series, please visit our website at:

www.atlantis-press.com/publications/books



AMSTERDAM – PARIS



© ATLANTIS PRESS / WORLD SCIENTIFIC

Multicore Systems On-Chip: Practical Software/Hardware Design

Abderazek Ben Abdallah

University of Aizu, Adaptive Systems Laboratory,
Aizuwakamatsu, 965-8580, Fukushima-ken, Japan



AMSTERDAM – PARIS



Atlantis Press

29, avenue Laumière
75019 Paris, France

For information on all Atlantis Press publications, visit our website at:

www.atlantis-press.com

Copyright

This book, or any parts thereof, may not be reproduced for commercial purposes in any form or by any means, electronic or mechanical, including photocopying, recording or any information storage and retrieval system known or to be invented, without prior permission from the Publisher.

Atlantis Computational Intelligence Systems

Volume 1: Agent-Based Ubiquitous Computing – Eleni Mangina, Javier Carbo, José M. Molina

Volume 2: Web-Based Information Technologies and Distributed Systems – Alban Gabilon, Quan Z. Sheng, Wathiq Mansoor

ISBN: 978-90-78677-22-2
ISSN: 1875-7669

e-ISBN: 978-94-91216-33-6

*To my wife Sonia,
my kids Tesnim and Beyram,
and my parents.*

Preface

To meet high computational demands posed by latest consumer electronic devices (PDAs, cell phones, laptops, cameras, etc.), current systems employ a multitude of multicore on a single chip. The attraction of multicore processing for power reduction is compelling. By splitting a set of tasks among multiple processor cores, we can reduce the operating frequency necessary for each core, allowing to reduce the voltage on each core. Because dynamic power is proportional to the frequency and to the square of the voltage, we get a big gain, even though we may have more cores running. Even static power improves as we turn down supply voltage. However, there are several barriers to designing general purpose and embedded multicore systems. Software development becomes far more complex due to the difficulties in breaking a single processing task into multiple parts that can be processed separately and then reassembled later. This reflects the fact that certain processor jobs cannot be easily parallelized to run concurrently on multiple processing cores and that load balancing between processing cores – especially heterogeneous cores – is very difficult. The other set of problems with multicore systems design are hardware-based.

The book consists of seven chapters. The first chapter introduces multicore system architecture and describes a design methodology for these systems. The architectures used in conventional methods of MCSoc design 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. This chapter will emphasize on the design techniques and methodologies

Power dissipation continues to be a primary design constraint 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. Energy conservation has been largely considered in the hardware design in general and also in embedded multicore system' components, such as CPUs, disks, displays, memories, and so on. Significant additional power savings can be also achieved by incorporating low-power methods into the design of network protocols used for data communication (audio, video, etc.). Chapter two investigates in details power reduction techniques at components and the network protocol levels.

Conventional on-chip communication design mostly use *ad-hoc* approaches that fail to

meet the challenges posed by the next-generation Multicore Systems-on-Chip (MCSoc) designs. These major challenges include wiring delay, predictability, diverse interconnection architectures, and power dissipation.

A Network-on-Chip (NoC) paradigm is emerging as the solution for the problems of interconnecting dozens of cores into a single system-on-chip. However, there are many problems associated with the design of such systems. These problems arise from non-scalable global wire delays, failure to achieve global synchronization, and difficulties associated with non-scalable bus-based functional interconnects.

In chapter three, we explain low-power and low-cost on-chip architectures in terms of router architecture, network topology, and routing for multi- and many-core systems.

To overcome challenges from high power densities and thermal hot spots in microprocessors, multi core computing platforms have emerged as the ubiquitous computing platform from servers to embedded systems. But, providing multiple cores does not directly translate into increased performance for most applications.

With the rise of multi-core systems and many-core processors, concurrency becomes a major issue in the daily life of a programmer. Thus, compiler and software development tools will be critical to help programmers create high performance software. Chapter four covers software issues of a so-called parallelizing queue compiler targeted for future single and multicore embedded systems.

Chapter five presents practical hardware design issues of a novel dual-execution mode processor (DEP) architecture targeted for embedded applications. Practical hardware design results and advanced optimization techniques are presented in a fair amount of details

Chapter six presents design and architecture of a produced order Queue core based on Queue computing and suitable for low power computing.

With the proliferation of portable devices, new multimedia-centric applications are continuously emerging on the consumer market. These applications are pushing computer architecture to its limit considering their demanding workloads. In addition, these workloads tend to vary significantly at run time as they are driven by a number of factors such as network conditions, application content, and user interactivity. Most current hardware and software approaches are unable to deliver executable codes and architectures to meet these requirements. There is a strong need for performance-driven adaptive techniques to accommodate these highly dynamic workloads. Chapter seven shows the potential of these techniques in both software and hardware domains by reviewing early attempts in dynamic binary translation on the software side and FPGA-based reconfigurable architectures on the hardware side. It puts forward a preliminary vision for unifying runtime adaptive techniques in hardware and software to tackle the demands of these new applications. This vision will not be possible to realize unless the notorious reconfiguration bottleneck familiar in FPGAs is addressed.

*Abderazek Ben Abdallah
The University of Aizu*

Contents

Preface	vii
List of Figures	xiii
List of Tables	xvii
1. Multicore Systems Design Methodology	1
1.1 Introduction	1
1.2 MCSoCs Design Problems	2
1.3 Multicore architecture platform	4
1.4 Application specific MCSoC design method	5
1.5 QueueCore architecture	6
1.5.1 Hardware pipeline structure	7
1.5.2 Floating point organization	8
1.6 QueueCore synthesis and evaluation results	12
1.6.1 Methodology	12
1.6.2 Design results	13
1.7 Conclusion	14
2. Design for Low Power Systems	17
2.1 Introduction	17
2.2 Power Aware Technological-level Design optimizations	19
2.2.1 Factors affecting CMOS power consumption	19
2.2.2 Reducing voltage and frequency	20
2.2.3 Reducing capacitance	21
2.3 Power Aware Logic-level Design Optimizations	22
2.3.1 Clock gating	22
2.3.2 Logic encoding	23
2.3.3 Data guarding	23
2.4 Power-Aware System Level Design Optimizations	24
2.4.1 Hardware system architecture power consumption optimizations	24

2.4.2	Operating system power consumption optimization	28
2.4.3	Application, compilation techniques and algorithm	29
2.4.4	Energy reduction in network protocols	30
2.5	Conclusion	34
3.	Network-on-Chip for Multi- and Many-Core Systems	35
3.1	Introduction	35
3.2	Router Architecture	37
3.2.1	Switching Technique	38
3.2.2	Router Components	39
3.2.3	Pipeline Processing	40
3.3	Topology	48
3.3.1	Interconnection Topologies	48
3.3.2	Topological Properties	52
3.4	Routing	55
3.4.1	Deadlocks and Livelocks	55
3.4.2	Routing Algorithm	56
3.5	Summary	59
4.	Parallelizing Compiler for High Performance Computing	61
4.1	Instruction Level Parallelism	61
4.2	Parallel Queue Compiler	64
4.3	Parallel Queue Compiler Frame Work	65
4.3.1	1-offset P-Code Generation Phase	66
4.3.2	Offset Calculation Phase	68
4.3.3	Instruction Scheduling Phase	70
4.3.4	Natural Instruction Level Parallelism Extraction: Statement Merging Transformation	71
4.3.5	Assembly Generation Phase	73
4.3.6	Results	74
4.4	Conclusion	78
5.	Dual-Execution Processor Architecture for Embedded Computing	79
5.1	Introduction	79
5.2	System Architecture	81
5.2.1	Pipeline Structure	82
5.2.2	Fetch unit	83
5.2.3	Decode unit	84
5.2.4	Dynamic switching mechanism	84
5.2.5	Calculation of produced and consumed data	85
5.2.6	Queue-Stack Computation unit	85
5.2.7	Sources-Results computing mechanism	86

5.2.8	Issue unit	89
5.2.9	Execution unit	91
5.2.10	Shared storage mechanism	92
5.2.11	Covop instruction execution mechanism	93
5.2.12	Interrupt handling mechanism	94
5.3	Sub-routine call handling mechanism	97
5.4	Hardware design and Evaluation results	100
5.4.1	DEP System pipeline control	101
5.4.2	Hardware Design Result	102
5.4.3	Comparison results	105
5.5	Conclusions	106
6.	Low Power Embedded Core Architecture	107
6.1	Introduction	107
6.2	Produced Order Queue Computing Overview	108
6.3	QC-2 Core Architecture	111
6.3.1	Instruction Set Design Considerations	111
6.3.2	Instruction Pipeline Structure	112
6.3.3	Dynamic Operands Addresses Calculation	114
6.3.4	QC-2 FPA Organization	115
6.3.5	Circular Queue-Register Structure	118
6.4	Synthesis of the QC-2 Core	119
6.4.1	Design Approach	119
6.5	Results and Discussions	121
6.5.1	Execution Speedup and Code Analysis	121
6.5.2	Synthesis Results	122
6.5.3	Speed and Power Consumption Comparison with Synthesizable CPU Cores	124
6.6	Conclusions	125
7.	Reconfigurable Multicore Architectures	127
7.1	Introduction	127
7.1.1	Performance-Driven Approaches in Software	128
7.1.2	Performance-Driven Approaches in Hardware	129
7.1.3	Potential of FPGA SOCs	130
7.2	Runtime Hardware Adaptation	130
7.2.1	Processor Array Architectures	131
7.2.2	Datapath Array Architectures	135
7.2.3	Single Processor Architectures	138
7.3	Summary of Hardware Adaptation	140
7.4	Runtime Software Adaptation	142
7.4.1	Warp Processing	143

7.4.2	Dynamic Instruction Merging	148
7.4.3	Summary of Software Adaptation	152
7.5	Future Directions in Runtime Adaptation	153
7.5.1	Future Hardware Adaptation	153
7.5.2	Future Software Adaptation	155
7.5.3	Future Reconfiguration Infrastructures in FPGA SOCs	156
7.6	Conclusion	157
	Bibliography	159
	Index	177

List of Figures

1.1	SoC typical architecture	3
1.2	MCSoc system platform	4
1.3	Linked-task design flow graph (DFG)	6
1.4	Next QH and QT pointers calculation mechanism	9
1.5	QC-2's source 2 address calculation	10
1.6	QC-2's FADD hardware	11
1.7	QC-2's FMUL hardware	12
1.8	Resource usage and timing for 256*33 bit QREG unit for different coding and optimization strategies	13
1.9	Achievable frequency is the instruction throughput for hardware implementations of the QC-2 processor	14
2.1	Clock gating example	23
2.2	Dual Operation ALU with Guard Logic	24
2.3	Power consumption in typical processor	27
2.4	Protocol stack of a generic wireless network, and corresponding areas of energy efficient possible research	32
3.1	Shared-media network	36
3.2	Switch-media network	36
3.3	Network-on-Chip	36
3.4	Wormhole router architecture	39
3.5	Router pipeline structure (4-cycle)	40
3.6	Router pipeline structure (3-cycle)	40
3.7	Router pipeline structure (2-cycle)	41
3.8	Prediction router architecture	43
3.9	Prediction router pipeline	43
3.10	Default-backup path mechanism in 16-core NoC	47
3.11	Two-dimensional layouts of typical topologies (1)	48
3.12	Two-dimensional layouts of typical topologies (2)	49
3.13	Deadlocks of packets	55

3.14	Prohibited turn sets of three routing algorithms in Turn-Model	56
3.15	Prohibited turn set in Odd-Even Turn-Model	57
4.1	Instruction sequence generation from the parse tree of expression $x = \frac{a+b}{b-c}$	62
4.2	Instruction sequence generation from DAG of expression $x = \frac{a+b}{b-c}$	62
4.3	QIR code fragment	71
4.4	Statement merging transformation	71
4.5	Assembly output for QueueCore processor	72
4.6	Effect on ILP of statement merging transformation in the queue compiler	76
4.7	Instruction level parallelism improvement of queue compiler over optimizing compiler for a RISC machine	77
4.8	Normalized code size for two embedded RISC processors and QueueCore	77
5.1	DEP architecture block diagram.	82
5.2	Block diagram of fetch unit.	83
5.3	Block diagram of decode unit.	84
5.4	Mode-switching mechanism.	85
5.5	Decode mechanism: (a) decode for queue program and (b) decode for stack program.	87
5.6	Block diagram of queue-stack computation unit.	88
5.7	Address calculation mechanism for sources and destination.	89
5.8	Address calculation mechanism for next instruction's source1 and destination.	90
5.9	Addresses calculation example: (a) QEM mode and (b) SEM mode.	91
5.10	Block diagram of issue unit.	92
5.11	Block diagram of execution unit.	92
5.12	Block diagram of shared storage unit.	93
5.13	Address extension mechanism.	93
5.14	Components used for software interrupt handling mechanism in DEP.	94
5.15	Queue status when interrupt occur and return from interrupt. (a) Queue status before interrupt, (b) when interrupt occur, (c) Queue: ready for interrupt handler, (d) Queue: when return from interrupt.	95
5.16	Stack status when interrupt occur and return from interrupt. (a) stack status before interrupt, (b) when interrupt occur, (c) stack: ready for interrupt handler, (d) stack: when return from interrupt.	96
5.17	Components used for subroutine call mechanism in DEP.	97
5.18	Queue status when subroutine call and return from call. (a) queue status before call, (b) when execute the call, (c) queue: ready for handle callee program, (d) when execute the return from call (rfc) instruction, and (e) queue: when return from call with return result.	98
5.19	Stack status when subroutine call and return from call. (a) stack status before call, (b) when execute the call, (c) stack: ready for handle callee program, (d) when execute the return from call (rfc) instruction, and (e) stack: when return from call with return result.	99

5.20	Finite state machine transition for DEP pipeline synchronization	101
5.21	Critical path for different units	103
6.1	Sample data flow graph and queue-register contents for the expressions: $e = ab/c$ and $f = ab(c + d)$. (a) Original sample program, (b) Translated (augmented) sample program, (c) Generated instructions sequence, (d) Circular queue-register content at each execution state.	109
6.2	QC-2 instruction format and computing examples: (a) <i>add</i> instruction, (b) <i>mod</i> instruction, (c) load immediate (<i>ldil</i>) instruction, (d) <i>call</i> instruction, and (e) store word (<i>stw</i>) instruction.	110
6.3	QC-2 architecture block diagram. During RTL description, the core is broken into small and manageable modules using modular approach structure for easy verification, debugging and modification.	112
6.4	Source 1 (<i>source1</i>) address calculation hardware.	113
6.5	Source 2 (<i>source2</i>) address calculation hardware.	114
6.6	QC-2's FPA Hardware: Adder Circuit	115
6.7	QC-2's FPA Hardware: Multiplier Circuit	116
6.8	Circular queue-register (QREG) structure. (a) initial QREG state; (b) QREG state after writing the first 32 bit data (dat1); (c) QREG state after writing the second data (dat2) and consuming the first 32 bit data (dat1); (d) QREG state with LQH pointer update and different regions.	117
6.9	Finite state machine transition for QC-2 pipeline synchronization. The following conditions are evaluated: next stage can accept data (ACP), previous pipeline stage can supply data (SUP), last cycle of computation (CPT).	118
6.10	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.	120
6.11	Resource usage and timing for 256*33 bit QREG unit for different coding and optimization strategies.	122
6.12	Floorplan of the placed and routed QC-2 core.	124
7.1	The three architectural levels in RAMPSoC	132
7.2	An example of a heterogeneous multiprocessor architecture template for a real-time image processing application	133
7.3	Mesh-based NOC	133
7.4	A multilayer reconfigurable NOC	134
7.5	Vertically oriented dynamic reconfigurable systems	134
7.6	A 2-dimensional dynamic reconfigurable system	135
7.7	System diagram of QUKU	136
7.8	Design flow of QUKU	136
7.9	Temporal and spatial mapping of an FIR filter on QUKU	137
7.10	Area and clock performance of QUKU, MicroBlaze, and a custom core for an FIR implementation	138

7.11	Logical view of the target architecture	139
7.12	HARPE processing element	139
7.13	Reconfiguration scenario	140
7.14	Hardware/software partitioning approaches	144
7.15	Overview of a warp processor	144
7.16	Overview of the ROCCAD tool chain	145
7.17	Overview of W-FPGA	146
7.18	Overview of the routing-oriented configurable logic fabric	147
7.19	Overall speedup by the warp processor and traditional hardware/software partitioning	147
7.20	Overall energy reduction by the warp processor and traditional hardware/software partitioning	148
7.21	Percentage of resources used to implement the benchmarks in the routing-oriented configurable logic fabric	148
7.22	DIM place in system architecture	149
7.23	A configuration example of the array	150
7.24	Energy consumption using a 64-slot cache with or without speculation	151
7.25	Performance and versatility vs. reconfiguration overhead	154
7.26	Diagram of a hardwired NOC on an FPGA SOC	158

List of Tables

1.1	Linked-task description	5
1.2	QC-2 processor design results: modules complexity as LE	13
2.1	Operating system functionality and corresponding techniques for optimizing energy utilization	28
3.1	Channel bisection B_c	52
3.2	Average hop count H_{ave}	52
3.3	Number of routers R	53
3.4	Total unit-length of links L (1-unit=distance between neighboring two cores)	54
4.1	Lines of C code for each phase of the queue compiler's back-end	74
4.2	Instruction category percentages for the compiled benchmarks for the QueueCore	74
4.3	QueueCore's program maximum offset reference value	75
5.1	PN and CN calculation with instruction in decode stage. PN means number of produced data and CN means number of consumed data	86
5.2	DEP Processor hardware configuration parameters	102
5.3	Verilog HDL code size for integrated DEP processor	103
5.4	Synthesis results. LEs means Logic Elements. AOP means Area optimization and SOP means speed optimization	104
5.5	Comparison results between DEP and PQP architecture. Area in LE, Speed in MHz, and Power in mW	104
5.6	DEP speed comparisons	105
5.7	DEP power consumption comparisons with various synthesizable CPU cores	105
6.1	Normalized code sizes for various benchmark programs over different target architectures	119
6.2	Execution time and speedup results	121
6.3	QC-2 Hardware Configuration Parameters	123

6.4	QC-2 processor design results: modules complexity as LE (logic elements) and TCF (total combinational functions) when synthesized for FPGAs (with Stratix device) and Structured ASIC (HardCopy II) families	123
6.5	Speed and power consumption comparisons for various Synthesizable CPU cores over speed (SPD) and area (ARA) optimizations. This evaluation was performed under the following constraints: (1) Family: Stratix; (2) Device: EP1S25F1020; (3) Speed: C6. The speed is given in MHz.	125
7.1	Configuration codes for the task graph of Figure 10	137
7.2	Comparison of configuration size and speed in QUKU and Montium . . .	138
7.3	Area cost of soft-core processors and dedicated cores	141
7.4	Summary of hardware architectural adaptation	141
7.5	Hardware configurations used in experimentation	150
7.6	Obtained speedups of the benchmarks over the three hardware configurations .	151
7.7	Area cost of the configurable array	152
7.8	Comparison of warp processing, DIM and the ideal BT	152
7.9	Overhead, impact on performance, frequency and usage for the three reconfiguration levels	155