Multi-core Architectures with Coarse-grained Dynamically Reconfigurable Processors for Broadband Wireless Access Technologies

Wei Han

A thesis submitted for the degree of Doctor of Philosophy.

The University of Edinburgh.

January 2010

Abstract

Broadband Wireless Access technologies have significant market potential, especially the WiMAX protocol which can deliver data rates of tens of Mbps. Strong demand for high performance WiMAX solutions is forcing designers to seek help from multi-core processors that offer competitive advantages in terms of all performance metrics, such as speed, power and area. Through the provision of a degree of flexibility similar to that of a DSP and performance and power consumption advantages approaching that of an ASIC, coarse-grained dynamically reconfigurable processors are proving to be strong candidates for processing cores used in future high performance multi-core processor systems.

This thesis investigates multi-core architectures with a newly emerging dynamically reconfigurable processor – RICA, targeting WiMAX physical layer applications. A novel master-slave multi-core architecture is proposed, using RICA processing cores. A SystemC based simulator, called MRPSIM, is devised to model this multi-core architecture. This simulator provides fast simulation speed and timing accuracy, offers flexible architectural options to configure the multi-core architecture, and enables the analysis and investigation of multi-core architectures. Meanwhile a profiling-driven mapping methodology is developed to partition the WiMAX application into multiple tasks as well as schedule and map these tasks onto the multi-core architecture, aiming to reduce the overall system execution time. Both the MRPSIM simulator and the mapping methodology are seamlessly integrated with the existing RICA tool flow.

Based on the proposed master-slave multi-core architecture, a series of diverse homogeneous and heterogeneous multi-core solutions are designed for different fixed WiMAX physical layer profiles. Implemented in ANSI C and executed on the MRPSIM simulator, these multi-core solutions contain different numbers of cores, combine various memory architectures and task partitioning schemes, and deliver high throughputs at relatively low area costs. Meanwhile a design space exploration methodology is developed to search the design space for multi-core systems to find suitable solutions under certain system constraints. Finally, laying a foundation for future multithreading exploration on the proposed multi-core architecture, this thesis investigates the porting of a real-time operating system – Micro C/OS-II to a single RICA processor. A multitasking version of WiMAX is implemented on a single RICA processor with the operating system support.

Declaration of originality

I hereby declare that the research recorded in this thesis and the thesis itself was composed by myself in the School of Engineering at The University of Edinburgh, except where explicitly stated otherwise in the text.

Wei Han

Acknowledgements

This thesis would not have been what you are seeing now, without the help of my supervisors and colleagues. Foremost, I would like to thank my PhD supervisors Prof. Tughrul Arslan and Dr. Ahmet T. Erdogan for their support and guidance during my study. I would also like to thank my colleagues, Dr. Ying Yi who made great contributions to our multi-core project and did a lot of work which I was too lazy to do, Mr. Mark I.R. Muir for his valuable guidance and parser libraries, Dr. Ioannis Nousias for his help on MRPSIM simulator design, Mr. Xin Zhao and Dr. Cheng Zhan for their contributions on WiMAX implementations. Meanwhile, I would like to thank all RICA team members: Dr. Sami Khawam, Dr. Mark Milward, Dr. Ioannis Nousias, Dr. Ying Yi and Mr. Mark I.R. Muir for their brilliant invention – the RICA architecture and its tool flow. In addition, many thanks to all members in SLI group for their help throughout my PhD study.

A very special thank to my wife Xinyu Chen who encourages me with her love, stays with me getting through all tough times and cooks delicious food for me. Finally, I would like to express my deepest appreciation to my parents for their love, guide and support to me throughout my life.

Acronyms and abbreviations

3G	Third Generation
ADSL	Asymmetric Digital Subscriber Line
AES	Advanced Encryption Standard
ALM	Adaptive Logic Module
ALU	Arithmetic Logic Unit
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
ASIP	Application-Specific Instruction Set Processor
ASSP	Application Specific Standard Product
BPSK	Binary Phase-Shift Keying
BS	Base Station
BWA	Broadband Wireless Access
CDMA	Code Division Multiple Access
CLB	Configurable Logic Block
CMP	Chip-Level Multiprocessing
CP	Cyclic Prefix
CPS	Cycles Per Second
DC	Direct Current
DL	Downlink
DSL	Digital Subscriber Line
DSP	Digital Signal Processor
DR	Dynamically Reconfigurable
EV-DO	Evolution-Data Optimized

FEC	Forward Error Correction
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FPGA	Field Program Gate Array
GF	Galois Field
GFLOPS	Giga Floating-Point Operations Per Second
GPP	General Purpose Processor
GSM	Global System for Mobile communications
GPU	Graphics Processing Unit
HDL	Hardware Description Language
HPT	Highest Priority Task
HSPA	High Speed Packet Access
HSDPA	High Speed Downlink Packet Access
HSUPA	High Speed Uplink Packet Access
IFFT	Inverse Fast Fourier Transform
ILP	Instruction Level Parallelism
IP	Intellectual Property
IPI	Inter-Processor Interrupt
IPS	Instructions Per Second
ISA	Instruction Set Architecture
ISDN	Integrated Services Digital Network
ISI	Intersymbol Interference
LL	Load-Link
LUT	Look-Up Table
MAC	Media Access Control
Mbps	Megabits per second
MIMO	Multi-Input Multi-Output
ML	Maximum Likelihood

MDF	Machine Description File
MRPSIM	Multiple Reconfigurable Processor Simulator
OFDM	Orthogonal Frequency-Division Multiplexing
OS	Operating System
PHY	Physical Layer
PLD	Programming Logic Device
PPE	Power Processing Element
PRBS	Pseudo-Random Binary Sequence
PSE	Processing and Storage Element
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase-Shift Keying
RA	Reconfigurable Architecture
RAM	Random Access Memory
RICA	Reconfigurable Instruction Cell Array
RISC	Reduced Instruction Set Computer
RRC	Reconfigurable Rate Controller
RPU	Reconfigurable Processing Unit
RS	Reed-Solomon
RTL	Register Transfer Level
RTOS	Real-Time Operating System
SC	Store-Conditional
SDL	System Description Level
SIMD	Single Instruction Multiple Data
SoC	System on Chip
SOFDMA	Scalable Orthogonal Frequency Division Multiple Access
SMP	Symmetric Multiprocessing
SPE	Synergistic Processing Element
SPMD	Single Program Multiple Data

SPS	Steps Per Second
SRAM	Static Random Access Memory
SS	Subscriber Station
TDD	Time Division Duplex
TLM	Transaction-Level modeling
TTA	Transport Triggered Architecture
UL	Uplink
UMTS	Universal Mobile Telecommunications System
VLIW	Very Long Instruction Word
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Network

Publication from this work

Journals

1. **W. Han**, Y. Yi, M. Muir, I. Nousias, T. Arslan, and A. T. Erdogan, "Multi-core Architectures with Dynamically Reconfigurable Array Processors for Wireless Broadband Technologies," *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems (TCAD)*, Vol. 28, Issue 12, pp. 1830-1843, December 2009.

2. **W. Han**, Y. Yi, M. Muir, I. Nousias, T. Arslan, and A. T. Erdogan, "Efficient Implementation of WiMAX Physical Layer on Multi-core Architecture with Dynamically Reconfigurable Processors," *Scalable Computing: Practice and Experience Scientific international journal for parallel and distributed computing*, Vol. 9, pp.185-196, ISSN 1097-2803, 2008.

Conference and Workshops

1. **W. Han**, Y. Yi, Xin Zhao, M. Muir, T. Arslan, and A. T. Erdogan, "Heterogeneous Multi-core Architectures with Dynamically Reconfigurable Processors for WiMAX transmitter", *the 22nd Annual IEEE International SOC Conference (SOCC'09*), pp. 97 - 100, Belfast, UK, 9 – 11 September, 2009.

2. **W. Han**, Y. Yi, Xin Zhao, M. Muir, T. Arslan, and A. T. Erdogan, "Heterogeneous Multi-core Architectures with Dynamically Reconfigurable Processors for Wireless Communication", *the 7th IEEE Symposium on Application Specific Processors in conjunction with Design Automation Conference 2008 (SASP'09*), pp. 1 - 6, San Francisco, California, 27 - 28 July, 2009.

3. Y. Yi, **W. Han**, X. Zhao, A. T. Erdogan and T. Arslan, "An ILP Formulation for Task Mapping and Scheduling on Multi-core Architectures," *the Conference on Design, Automation and Test in Europe (DATE'09)*, pp. 33-38, Nice, France, 20 – 24 April, 2009.

4. **W. Han**, Y. Yi, M. Muir, I. Nousias, T. Arslan, and A. T. Erdogan, "MRPSIM: a TLM based Simulation Tool for MPSoCs targeting Dynamically Reconfigurable Processors," *the 21st Annual IEEE International SOC Conference (SOCC'08)*, pp. 41-44, Newport Beach, California, 17 - 20 September, 2008.

5. Y. Yi, **W. Han**, A. Major, A. T. Erdogan, and T. Arslan, "Exploiting Loop-Level Parallelism on Multi-Core Architectures for the WiMAX Physical Layer," *the 21st Annual IEEE International SOC Conference (SOCC'08)*, pp. 31-34, Newport Beach, California, 17 – 20 September, 2008.

6. **W. Han**, Y. Yi, M. Muir, I. Nousias, T. Arslan, and A. T. Erdogan, "Multi-core Architectures with Dynamically *Reconfigurable Array Processors for the WIMAX Physical*

Layer," the 6th IEEE Symposium on Application Specific Processors conjunction with Design Automation Conference 2008 (SASP'08), pp.115-120, Anaheim, California, 8 – 9 June, 2008.

7. **W. Han**, Y. Yi, M. Muir, I. Nousias, T. Arslan, and A. T. Erdogan, "Efficient Implementation of Wireless Applications on Multi-core Platforms based on Dynamically Reconfigurable Processors," *2008 International Workshop on Multi-Core Computing Systems*, pp. 837-842, Barcelona, Spain, March 2008.

8. **W. Han**, I. Nousias, M. Muir, T. Arslan, A.T. Erdogan, "The Design of Multitasking Based Applications on Reconfigurable Instruction Cell Based Architectures," *the 17th International Conference on Field Programmable Logic and Applications (FPL'07)*, pp. 447-452, Amsterdam, Netherlands, 27 – 29 August, 2007.

9. **W. Han**, M. Muir, I. Nousias, T. Arslan, A.T. Erdogan, "Mapping Real Time Operating System on Reconfigurable Instruction Cell based Architectures," *2007 IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'07)*, pp.301-304, Napa, California, 23 – 25 April, 2007.

Contents

1 Introduction ……………………………………………………………………..…1
1.1 Motivation 1
1.2 Objective 3
1.3 Contribution 4
1.4 Thesis structure........ 5

2 Broadband Access Technologies and WiMAX…...…………………………….………7
2.1 Introduction 7
2.2 Wired broadband access technologies 8
2.2.1 DSL 8
2.2.2 Cable 9
2.2.3 Optical fibre 9
2.2.4 Others 9
2.3 Broadband wireless access technologies 10
2.3.1 HSPA 10
2.3.2 EV-DO 11
2.3.3 Satellite 11
2.3.4 Wi-Fi 12
2.3.5 WiMAX 12
2.4 WiMAX PHY processing chain 14
2.4.1 Randomising 15
2.4.2 FEC encoding 15
2.4.2.1 Reed-Solomon encoding 16
2.4.2.2 Convolutional encoding 17
2.4.2.3 Puncturing 18
2.4.3 Interleaving 19
2.4.4 Data modulation........ 20
2.4.5 OFDM processing........ 21
2.4.5.1 Pilot modulation and insertion 22
2.4.5.2 DC and guard band insertion........ 23
2.4.5.3 FFT/IFFT 23
2.4.5.4 Cyclic prefix extension 24
2.4.6 Synchronisation 25
2.4.7 Data demodulation........ 26

2.4.8 FEC decoding....26
2.4.8.1 Viterbi decoding....26
2.4.8.2 Reed-Solomon decoding....27
2.5 Summary....28

3 Multi-core Processors and Reconfigurable Architectures....29
3.1 Introduction....29
3.2 Multi-core processors....30
3.2.1 The advantages of multi-core processors....31
3.2.2 Taxonomies of multi-core processors....32
3.2.2.1 Computing markets....32
3.2.2.2 Heterogeneity and Homogeneity....33
3.2.2.3 Core types....35
3.3 Reconfigurable architectures....36
3.3.1 Execution structures....36
3.3.2 Granularity....37
3.3.2.1 Fine-grained reconfigurable architectures....37
3.3.2.2 Coarse-grained reconfigurable architectures....38
3.3.2.3 Medium-grained and very coarse-grained reconfigurable architectures....40
3.3.2.4 Mixed-grained reconfigurable architectures....40
3.3.3 Reconfiguration schemes....41
3.4 Reconfigurable instruction cell array....41
3.4.1 Architecture....42
3.4.2 Standard tool flow....44
3.5 Traditional WiMAX silicon implementations....45
3.5.1 Custom chip implementations....45
3.5.2 GPP/DSP based SoC implementations....46
3.5.3 FPGA/PLD implementations....47
3.6 Multi-core implementations of WiMAX....47
3.6.1 PicoArray....48
3.6.2 Freescale MSC8126....49
3.6.3 Cell Broadband Engine....49
3.6.4 Intel IXP 2350 network processor....50
3.6.5 Sandbridge SB3010....50
3.7 RICA based multi-core architecture....51
3.8 Summary....51

4 A RICA Processor based Multi-core Architecture....53
4.1 Introduction....53
4.2 The proposed multi-core architecture....54
4.2.1 Processing cores....56
4.2.2 Memory architecture....57
4.2.2.1 Shared memory/local memory....57

4.2.2.2 Shared register file 58
4.2.2.3 Stream buffer 60
4.2.3 Arbiter 62
4.2.4 Crossbar switch 62
4.2.5 Interrupt controller 63
4.2.6 Router 63
4.3 Synchronisation methods and atomic operation support 64
4.4 Custom instruction integration 66
4.5 Summary 67

5 Multiple Reconfigurable Processors Simulator 69
5.1 Introduction 69
5.2 Related work on multiprocessor simulators 70
5.3 Trace-driven simulation 71
5.4 SystemC and transaction-level modeling 74
5.5 TLM model 76
5.6 MRPSIM command line options 79
5.7 Mpsockit - a subsidiary tool 81
5.8 Results 82
5.9 Summary 84

6 Homogeneous Multi-core Solutions for WiMAX 87
6.1 Introduction 87
6.2 Mapping methodology 88
6.3 Task partitioning methods 90
6.3.1 Task merging and task replication 91
6.3.2 Loop-level partitioning 93
6.4 Development of multi-core projects 96
6.5 Synchronisation between tasks 97
6.6 Results 98
6.7 Summary 104

7 Heterogeneous Multi-core Solutions for WiMAX 105
7.1 Introduction 105
7.2 Design space exploration 106
7.3 Timing optimisation 111
7.3.1 Code optimisation 111
7.3.2 Architectural optimisation 113
7.4 Area optimisation 114
7.5 Results 115
7.6 Summary 123

8 Multitasking WiMAX on an RTOS 125

8.1 Introduction 125
8.2 The selection of RTOSes 127
8.3 Porting Micro C/OS-II to RICA 130
8.3.1 Hardware requirements 130
8.3.2 Software requirements 131
8.4 The multitasking implementation of WiMAX on Micro C/OS-II 132
8.5 Results 136
8.6 Summary 138

9 Conclusions........ 141
9.1 Introduction 141
9.2 Review of thesis contents 141
9.3 Specific findings 144
9.4 Directions for future work 146

Reference 149

List of Figures

Figure 2.1 Fixed WiMAX physical layer processing chain 15
Figure 2.2 PRBS for the data randomisation 15
Figure 2.3 Reed-Solomon encoder 17
Figure 2.4 Binary convolutional encoder with a constraint length of 7 and a code rate of 1/2 17
Figure 2.5 Trellis diagram representation of a simple convolutional encoder 18
Figure 2.6 BPSK, QPSK, 16QAM and 64QAM modulation schemes in WiMAX 20
Figure 2.7 Frequency domain representation of an OFDM symbol in fixed WiMAX 22
Figure 2.8 OFDM downlink processing in the transmitter side of fixed WiMAX 22
Figure 2.9 PRBS for pilot modulation 23
Figure 2.10 Data flow graph of an 8-point radix-2 FFT 24
Figure 2.11 Signal flow representation of a radix-2 butterfly 24
Figure 2.12 Timing synchronisation 25
Figure 2.13 16QAM soft decision data demodulation 26
Figure 2.14 Viterbi decoding process for a convolutional code with a 1/2 code rate 27
Figure 2.15 Reed-Solomon decoding process 28
Figure 3.1 The RICA architecture 43
Figure 3.2 Standard RICA tool flow 45
Figure 3.3 PicoArray interconnection and the inter-processor communication 48
Figure 4.1 Proposed master-slave multi-core architecture 55
Figure 4.2 Custom instruction cell MULTIPTBK_REG_FILE 59
Figure 4.3 A 4-bank shared register file in the proposed multi-core architecture 60
Figure 4.4 Stream buffers in the proposed multi-core architecture 61
Figure 4.5 RICA custom cell generation environment and custom tool flow 67
Figure 5.1 The interface and internal structure of MRPSIM simulator 72
Figure 5.2 Traditional SoC design flow without SystemC 74
Figure 5.3 SoC design flow with SystemC 75
Figure 5.4 An example demonstrating the TLM model of a multi-core architecture 77
Figure 5.5 The simulation speed (steps/sec) 83
Figure 5.6 The simulation speed (instructions/sec) 83
Figure 5.7 The simulation speed (cycles/sec) 84
Figure 5.8 The simulation time (sec) 85
Figure 6.1 Mapping methodology 89
Figure 6.2 Task merging and task replication methods 92
Figure 6.3 Decoding loop partitioning 95

Figure 6.4 Integration of MRPSIM in Eclipse 96
Figure 6.5 Speedup and parallel efficiency of transmitter scenarios 101
Figure 6.6 Speedup and parallel efficiency of receiver scenarios 102
Figure 7.1 The main blocks in 16QAM based WiMAX transmitter and receiver 110
Figure 7.2 Task partitioning, mapping and scheduling in receiver Design 4 121
Figure 8.1 The Micro C/OS-II based system 130
Figure 8.2 Hardware requirements for porting Micro C/OS-II to RICA 131
Figure 8.3 Multitasking WiMAX function blocks 132
Figure 8.4 Program flow chart of WiMAX running on Micro C/OS-II 133
Figure 8.5 Detailed program flow diagram for the execution of channel coding and OFDM downlink tasks 134

List of Tables

Table 2.1 Pros and cons of wired and wireless broadband access technologies.............................. 7
Table 2.2 Comparison of 802.11 standards 12
Table 2.3 Comparison of broadband Internet access technologies.. 13
Table 2.4 Comparison between Fixed and Mobile WiMAX... 14
Table 2.5 Puncturing patterns in WiMAX.. 19
Table 3.1 Comparison of multi-core processors.. 35
Table 3.2 Comparison of coarse-grained reconfigurable architectures 39
Table 3.3 RICA Instruction Cells 42
Table 3.4 MSC8126 loading and task allocation... 49
Table 4.1 Access modes and configuration bits for RMEM and WMEM cells............................. 57
Table 4.2 Pin descriptions of MULTIPTBK_REG_FILE cells 59
Table 4.3 Access modes and configuration bits of MULTIPTBK_REG_FILE cells 59
Table 4.4 Pin description of SBUF instruction cells 61
Table 4.5 Access modes and configuration bits of SBUF cells 61
Table 4.6 Access modes and configuration bits for LL/SC instructions.. 66
Table 5.1 Functionality of main SystemC processes and functions in MRPSIM modules 78
Table 5.2 Command line options of MRPSIM.. 80
Table 5.3 Formats for original and compressed traces.. 81
Table 5.4 Mpsockit options 82
Table 6.1 IFFT Loop Partitioning.. 94
Table 6.2 Synchronisation between tasks.. 97
Table 6.3 Homogeneous multi-core scenarios for WiMAX transmitter.. 98
Table 6.4 Homogeneous multi-core scenarios for WiMAX receiver 98
Table 6.5 Configurations of both a standard core and a custom core.. 99
Table 6.6 Performance comparison for transmitter scenarios 102
Table 6.7 Performance comparison for receiver scenarios.. 103
Table 7.1 Pseudo code of the design space exploration methodology 108
Table 7.2 A code example of replacing jumps with multiplexers.. 111
Table 7.3 Storing constants in registers for 16QAM modulation.. 112
Table 7.4 Comparison of transmitter design configurations.. 116
Table 7.5 Comparison of receiver design configurations 116
Table 7.6 Comparison between standard and custom RICA cores used in transmitter designs .. 117
Table 7.7 Comparison between standard and custom RICA cores used in receiver designs....... 118
Table 7.8 Results from the exploration methodology for transmitter designs............................. 119

Table 7.9 Results from the exploration methodology for receiver designs 119
Table 7.10 Comparison of multi-core solutions 122
Table 8.1 Survey of RTOSes 127
Table 8.2 Comparison in term of memory size and multiprocessing support 129
Table 8.3 Comparison in term of ported devices and specific features 129
Table 8.4 Micro C/OS-II machine dependent files for RICA.. 131
Table 8.5 Results of WiMAX with/without RTOS support.. 136
Table 8.6 Comparison of multitasking WiMAX on RICA and ARM7TDMI 137

Chapter 1
Introduction

1.1 Motivation

Growing from an academic tool to owning almost one and half billion users nowadays, the Internet stands as a pivotal success. People have come to depend on the Internet more and more in their daily lives. Now users not only need traditional applications like Web surfing and emails, but also demand to experience multimedia services like interactive gaming, real-time audio and video streaming, and even high-definition TV. These applications are becoming an integral part of business as well as recreation, leading to the increased push for broadband access technologies with higher data rates, better reliability and enhanced user experience. Meanwhile currently the desire to access the Internet at any time from anywhere is dramatically rising. By combining the convenience through wireless access and the outstanding broadband performance, Broadband Wireless Access (BWA) technologies demonstrate great interests to end users and a remarkable potential success in commerce. Among various BWA solutions, Worldwide Interoperability for Microwave Access (WiMAX) defined in IEEE 802.16 standard family [1, 2] becomes more and more popular and may help make the vision of pervasive connectivity a reality. According to the announcement from the WiMAX forum, more than 133 million WiMAX users are anticipated by 2012 [3]. Meanwhile, the WiMAX market is forecasted to reach $3.5 billion and 4% market share of the overall broadband market by 2010 [3].

As an embedded application, WiMAX demands high throughput, strict low power, and in-field reprogrammability in order to follow evolving standards. Single-core processors fall short of meeting all these requirements. Historically, processor manufacturers answered the requirement for more processing power primarily by delivering faster processor speeds. The technology advances, such as Instruction Level Parallelism (ILP) and increased frequency due to advanced process technology, have delivered exponential performance gains for microprocessors over the last three decades. However things have changed and these traditional approaches will not provide the same gains in the future. It is mainly because high performance single-core microprocessors have reached three walls. Firstly, the memory wall results from the fact that the memory performance has a much slower growth rate than the processor performance. Therefore the gap between memory and processor speeds keeps increasing. In spite of the use of larger caches, this trend has not been curbed [4]. As a result, high frequency processors have to spend a significant amount of time waiting for the response from memory. The second wall comes from the limitation of ILP, called ILP wall. Even with various ambitious hardware and software techniques, only very low average parallelism can be found in the instruction streams and is far from enough to keep high performance processors busy [5]. The final wall is the power wall which is the main challenge incurred by high clock rates and deep submicron processes. In fact, in traditional single-core architectures, power consumption has increased at a greater rate than the clock speed. Meanwhile high processor density in sub-90 nm processes results in high power density which has already reached the level of 100W/cm^2 found in a nuclear reactor and probably soon will rise to the level of 1,000W/cm^2 found in space rocket nozzles [6].

These three walls drive people to look for alternative solutions, instead of continuing to build complicated single-core processors. One of such scale-out solutions is to put two or more simpler and smaller processing cores on a chip. It is called Chip-Level Multiprocessing (CMP) or a multi-core processor. In multi-core solutions, multiple simple processing cores with lower operating frequency can deliver excellent overall performance while reducing the thermal output. Another drive for the adoption of multi-core processors is that parallelism exists in most real world applications such as multimedia processing and Fast Fourier Transform (FFT). As a parallel architecture, a multi-core architecture can effectively bring the parallel nature of applications into full play.

However one question arising from multi-core architectures is that what kind of processing core would be the best candidate for multi-core architectures targeting embedded applications like WiMAX? Obviously, General Purpose Processors (GPPs) are not well suited to this task, due to their generic features and proportion of the silicon for computation. On the other hand, Very Long Instruction Word (VLIW) Digital Signal Processor (DSP) architectures face the problem due to the limited amount of ILP found in programs [7]. Filling the gap between the high flexibility of DSPs and the high performance of Application-Specific Integrated Circuits (ASICs), Dynamically Reconfigurable (DR) processors offer an attractive solution for developing multi-core architectures. This thesis investigates multi-core processor architectures building upon coarse-grained dynamically reconfigurable processing cores to meet the rigorous requirements from BWA applications, such as WiMAX.

1.2 Objective

The objective of this thesis is to explore a multi-core architecture by using coarse-grained dynamically reconfigurable processing cores, and develop tools and a mapping methodology which allow sequential ANSI C programs to be executed on the multi-core architecture. Afterward, based on this multi-core architecture, this thesis aims to design efficient multi-core solutions for broadband wireless access technologies, particularly WiMAX applications.

1.3 Contribution

The major contributions of the thesis are split into seven key aspects:

1. A novel master-slave multi-core architecture using a newly emerging coarse-grained DR processor, Reconfigurable Instruction Cell Array (RICA), has been proposed. This architecture provides a variety of memory architectural options for different application requirements and system constraints, as well as supporting inter-processor synchronisation through providing atomic operations.

2. A SystemC Transaction-Level modeling (TLM) based trace-driven simulator has been designed for modeling this multi-core architecture. This simulator delivers high-speed simulation and maintains timing accuracy. By providing flexible architectural options to configure the multi-core architecture, this simulator enables fast analysis and exploration of multi-core architectures as well as rapid application verification.

3. A profiling-driven mapping methodology has been developed to partition a target application into multiple tasks as well as schedule and map these tasks onto multi-core architectures, aiming to reduce the overall system execution time. This mapping methodology supports both homogeneous and heterogeneous multi-core solutions.

4. Several homogenous multi-core solutions have been developed for the Binary Phase-Shift Keying (BPSK) based fixed WiMAX physical layer application. These multi-core solutions combine different memory architectures and task partitioning schemes, and deliver high speedups compared to single-core implementations.

5. A design space exploration methodology has been presented to find suitable single-core and multi-core solutions under certain system constraints. Furthermore, several

timing and area optimisation techniques have been addressed for WiMAX applications, targeting the proposed multi-core architecture.

6. Several heterogeneous multi-core solutions have been developed for the 16-Quadrature Amplitude Modulation (QAM) based fixed WiMAX physical layer application. These solutions deliver high throughputs at relatively low area costs.

7. A Real-Time Operating System (RTOS) – Micro C/OS-II has been ported to a single-core RICA processor. A multitasking design of a fixed WiMAX physical layer program has been implemented on RICA processor with this operating system support.

1.4 Thesis structure

This thesis is structured as follows.

Chapters 2 and 3 contain descriptions of the background and existing literature. Chapter 2 provides an overview of mainstream broadband Internet access technologies and detailed descriptions for a fixed WiMAX physical layer. Chapter 3 introduces multi-core processors and reconfigurable computing technologies, and emphasises a dynamically reconfigurable processor – RICA, which is used in the proposed multi-core architecture. Meanwhile, Chapter 3 presents existing WiMAX implementations on various technologies.

Chapters 4 through 8 address my PhD research achievements. Chapter 4 proposes a basic multi-core architecture containing RICA based processing cores, and details the main components of this architecture as well as the inter-processor synchronisation methods. Chapter 5 presents a SystemC TLM trace-driven simulator which models the multi-core architecture described in Chapter 4. In addition, a trace preprocessing tool – Mpsockit is described in Chapter 4. Chapter 6 introduces homogeneous multi-core solutions for WiMAX, based on the proposed multi-core architecture. A mapping methodology and several task partitioning methods are described in this chapter as well. Chapter 7 focuses on heterogeneous multi-core implementations for WiMAX. A design space exploration methodology is presented to find suitable single-core or multi-core solutions under specific system and performance constraints. Chapter 8 investigates the porting of a real-time operating system to RICA processor and a multitasking based WiMAX on a single-core RICA processor with an operating system supported.

Finally, the thesis is concluded with the summary in Chapter 9.

Chapter 2
Broadband Access Technologies and WiMAX

2.1 Introduction

As broadband services are frequently upgraded, the minimum bandwidth or data rate for defining broadband is gradually increased. Generally speaking, nowadays a broadband access technology should provide data rates beyond 1 Megabits per second (Mbps). Broadband access technologies can be classified into wired broadband access and broadband wireless access. Both the two categories have their own pros and cons shown in Table 2.1. In some places, they are complementary, while in other places, they compete with each other.

Table 2.1 Pros and cons of wired and wireless broadband access technologies

	Pros	Cons
BWA	Provide portable, nomadic, mobile service; Serve extremely wide areas; Good solution for areas lacking wireline infrastructures	Lower capacity than wired broadband; Less reliable and secure
Wired broadband	High capacity at high data rates High reliability and security	Expensive to deploy new networks, especially in places lacking infrastructures

As currently the desire to access the Internet at any time from anywhere dramatically increased, BWA technologies demonstrate remarkable advantages over their wire based counterparts in many emerging markets such as the burgeoning mobile Internet market where wired broadband solutions are helpless. Meanwhile BWA technologies are gradually taking more market share from traditional markets such as the last mile delivery. Here the term of last mile represents the final step in the connection from Internet service providers to end users, and does not really mean that the actual distance is one mile.

In this chapter, a brief overview is provided for both wired and wireless broadband access technologies in Sections 2.2 and 2.3, respectively. Mainstream broadband Internet access technologies are reviewed. In Section 2.4, an introduction is given for WiMAX physical layer with the details of each functional block. The aim of Section 2.4 is to present the background and context necessary for understanding the WiMAX physical layer.

2.2 Wired broadband access technologies

By delivering Internet services through physical wires (e.g. twisted pair and television cable), wired broadband access technologies have inherent advantages over their wireless counterparts in reliability and security. Nowadays, there exist various wired technologies, some of which such as Digital Subscriber Line (DSL) and cable are predominant technologies in the market. The following subsections describe mainstream wired technologies available in the commercial market.

2.2.1 DSL

DSL is the most popular technology for the last mile delivery. According to the world broadband statistics report [8], almost 65 percent of wired broadband subscribers are using DSL by Q2 2008. Actually, DSL is a family of technologies transmitting data over the high frequency band of telephone lines, while keeping voice delivered over the low frequency band. Among DSL technologies, Asymmetric Digital Subscriber Line (ADSL) is the most common type used for home users and offers a much faster speed in the download direction than the upload direction. The latest ADSL standard provides data rates up to 24 Mbps and 3.5 Mbps for download and upload, respectively [9]. One of main drawbacks of DSL technologies is that data rates deteriorate with the distance from customers to providers' facilities increasing.

2.2.2 Cable

Cable Internet is a technology to enable Internet access over the existing cable television network, especially popular in North America. According to the world broadband statistics report [8], cable Internet has up to 82 million users, more than 55 percent of them from North America. Unlike DSL which is distance sensitive, cable can maintain the same data rate over 100 kilometres. As defined in an international standard DOCSIS 2.0 [10], cable Internet can provide maximum data rates of 50 Mbps and 27 Mbps for download and upload, respectively.

2.2.3 Optical fibre

Currently, many Internet service providers especially the ones in U.S. are using the optical fibre to deliver last mile communication, by directly connecting fibre to subscribers' premises. This fibre to the premises service includes the forms of fibre to the home and fibre to the building [11]. Depending on areas and communication protocols, the fibre to the premises offers data rates ranging from several Mbps to over one hundred Mbps. One disadvantage of this technology is the much more expensive connection charge and annual rental compared to other Internet access methods.

2.2.4 Others

Other wired broadband technologies include Integrated Services Digital Network (ISDN), power line, T1/E1 and so on. Like DSL, ISDN is based on the telephone network. It was popular prior to the emergence of DSL and cable. Now ISDN is gradually being superseded by DSL. Broadband over power line is a technology aiming to provide a broadband Internet at a data rate of over 100 Mbps via electric power lines. IEEE P1901 [12] is a draft standard for specifying this technology. During the thesis writing, this standard is still waiting for the approval. As for T1/E1, it can provide up to 2 Mbps bandwidth and is used for leased lines which directly connect customers such as companies to remote exchanges.

2.3 Broadband wireless access technologies

Although currently wired broadband access technologies can offer better reliability and security, BWA technologies have very close performance in these fields. More importantly, BWA systems enable mobility and ubiquitous Internet access, especially in areas where their wired counterparts are not available. Since the first generation BWA systems (e.g. local multipoint distribution systems and multichannel multipoint distribution services) were deployed in the late 1990s, many BWA technologies have emerged with higher data rates, longer ranges and better reliability. They include Code Division Multiple Access (CDMA) based High Speed Packet Access (HSPA) [13] and Evolution-Data Optimized (EV-DO) [14] as well as Orthogonal Frequency-Division Multiplexing (OFDM) based Wi-Fi [15] and WiMAX. In the following subsections, mainstream BWA technologies are introduced.

2.3.1 HSPA

HSPA is a family of protocols to improve the data transfer speeds in Universal Mobile Telecommunications System (UMTS). UMTS is one of Third Generation (3G) technologies, and combines wideband CDMA air interface and popular Global System for Mobile communications (GSM) infrastructures. HSPA delivers data through the UMTS network, and hence enables mobile broadband by using UMTS handheld devices. HSPA mainly consists of two standards, High Speed Downlink Packet Access (HSDPA) and High Speed Uplink Packet Access (HSUPA). Using approaches such as fast scheduling and link adaptation, HSDPA offers a maximum data rate of 14.4 Mbps for the user downloading, while its uploading complement HSUPA provides an uploading speed up to 5.76 Mbps through the help of methods like an enhanced dedicated physical channel [16].

2.3.2 EV-DO

EV-DO was designed to improve the data rates of another 3G technology, CDMA2000. Defined in [14], a combination of up to fifteen 1.25 MHz channels can boost the theoretical maximum rate of EV-DO to 73 Mbps. However, a typical deployment would involve just 3 channels to provide a download data rate of 14.7 Mbps. By using many of the same optimisation technologies as HSPA, EV-DO can achieve a similar spectral efficiency as HSPA. While one disadvantage of EV-DO is that EV-DO can not satisfy both voice and high-speed data communication concurrently, since it uses just one separate narrower channel for data service, compared to the efficient resource allocation for voice and data in a 5MHz channel used in HSPA. Moreover EV-DO has much less users than HSPA, as the UMTS network which HSPA is based on has more than nine times the subscribers of CDMA2000 [16].

2.3.3 Satellite

Satellite broadband is a wireless broadband access method which transmits and receives data via satellites in the geostationary orbit. There are two types of this method, one is one-way communication where the uploading and the downloading are through telephone lines and satellites, respectively. The other is two-way communication which enables both uploading and downloading via satellites. Satellite broadband is suitable for locations where other Internet access methods are not available, such as the remote rural areas and vessels at sea. The main drawback of this method compared to others is the much higher latency due to the very long signal trip. In one-way communication, the signal travel path is 70,000 km, while two-way communication has a path of 140,000 km. Meanwhile, satellite broadband is more likely affected by the weather and obstacles. Furthermore, the devices for satellite broadband are highly expensive [17].

2.3.4 Wi-Fi

Recently, Wireless Local Area Network (WLAN) technologies, especially Wi-Fi, have become popular. Nowadays, almost every desktop or laptop computer is shipped with Wi-Fi functions, and varied handheld devices such as smart phones have built-in Wi-Fi chips. Moreover many cities have developed or are developing a municipal Wi-Fi project to enable a whole city or at least the city centre Internet accessible through setting up hundreds of Wi-Fi access points. Wi-Fi technologies are specified in IEEE 802.11 standards [15], four main versions of which are compared in Table 2.2. Even though the popularity of Wi-Fi, compared to other BWA technologies, Wi-Fi has some intrinsic drawbacks such as too short range, not supporting mobility and inefficient collision avoidance protocols [18].

Table 2.2 Comparison of 802.11 standards

Standard	Release year	Operating frequency	Data rates (Typical/Max)	Range (Indoor/Outdoor)
802.11a	1999	5 GHz	23/54 Mbps	~35/120 m
802.11b	1999	2.4 GHz	4.5/11 Mbps	~38/140 m
802.11g	2003	2.4 GHz	19/54 Mbps	~38/140 m
802.11n	Nov 2009	2.4/5 GHz	74/600 Mbps	~70/250 m

2.3.5 WiMAX

Being the commercial name of IEEE 802.16 family of standards, WiMAX can provide up to tens of Mbps symmetric bandwidth over many kilometres. This gives WiMAX a significant advantage over other alternative last mile technologies like Wi-Fi and DSL. Based on OFDM, WiMAX can offer higher peak and average data rates, greater flexibility and system capacity, compared to CDMA based technologies, such as HSPA and EV-DO [18]. Table 2.3 provides a comparison between WiMAX and other broadband access technologies

Table 2.3 Comparison of broadband Internet access technologies

Technology	Access method	Max DL/UL (Mbps)	Range
DSL	wired	24/4.6	3 km
Cable	wired	50/27	160 km
Fibre	wired	50/20	20 km
HSPA	wireless	14.4/5.8	40 km
EV-DO	wireless	14.7/5.4	50-70 km
Satellite	wireless	16/2	35,000 km
Wi-Fi	wireless	54/54	250 m
WiMAX	wireless	70/70	50 km

described in previous sections, in terms of Downlink/Uplink (DL/UL) data rates and the coverage. In this context, the downlink represents the transmission path from a Base Station (BS) to a Subscriber Station (SS), and the uplink represents the converse transmission path. In Table 2.3, for wired broadband access technologies, the range represents the typical distance between the subscribers' devices to a DSL access multiplexer, a cable model termination system or a fibre local central office. Among these technologies, WiMAX offers the highest symmetric data rates on downlink and uplink, as well as very good coverage.

Usually, WiMAX standards offer a wide variety of options, such as a selectable channel bandwidth and spectrum profiles, which allow the industry to set up very flexible deployments to satisfy requirements from different markets. Due to this flexibility, the WiMAX Forum was formed to certify and promote interoperable industry solutions based on the IEEE 802.16 standards. Currently there are more than 400 certified WiMAX trial or commercial networks developed around the world. One such example is WiBro developed in South Korea. The IEEE 802.16 standard family includes two major standards, 802.16-2004 (i.e. 802.16d) [2] and 802.16-2005 (i.e. 802.16e) [1] which define fixed WiMAX and mobile WiMAX, respectively. Table 2.4 provides a comparison between the two standards in some basic characteristics. It can be seen that both fixed WiMAX and mobile WiMAX have their own advantages and target applications.

This thesis focuses on the fixed WiMAX technology defined in IEEE 802.16-2004 standard.

Table 2.4 Comparison between Fixed and Mobile WiMAX

Feature	Fixed WiMAX (802.16-2004)	Mobile WiMAX (802.16-2005)
Status	Completed June 2004	Completed December 2005
Spectrum	2-11 GHz	2-6 GHz
Application	Wireless DSL and Backhaul	Mobile Internet
Transmission Scheme	256-sub-carriers OFDM	Scalable Orthogonal Frequency-division Multiple Access (SOFDMA)
Service	Fixed and portable	Nomadic, portable and mobile
Typical Cell Radius	4-6 miles	1-3 miles

Fixed WiMAX is much likely to succeed in three Internet service markets: residential area and small or medium enterprises, leased lines for business as well as the backhaul for Wi-Fi hotspots [18]. The first category of market demands services like high-speed file downloading, voice over Intellectual Property (IP) and multimedia applications. For this market, fixed WiMAX based broadband services can be provided by either installing an outdoor antenna or an indoor cable-like modem. Therefore, fixed WiMAX is superior to wired based cable and DSL in terms of faster and easier installation, lower deployment and operational costs. The second potential market is the leased line connecting large companies directly to remote exchanges. In this field, fixed WiMAX is good enough to compete with the dominant technology - T1/E1 by providing higher speed services for enterprise customers which are not able to access fibre. Another major opportunity for fixed WiMAX is to serve as a faster and more cost-efficient alternative to the wired backhaul technology for Wi-Fi hotspots. Meanwhile fixed WiMAX is technically and commercially viable for the area lack of good wired infrastructures, such as developing countries and rural areas in developed countries.

2.4 WiMAX PHY processing chain

In this thesis, the target application is the fixed WiMAX Physical Layer (PHY) including both transmitter and receiver. Figure 2.1 provides an overview of the digital domain blocks in a typical fixed WiMAX PHY.

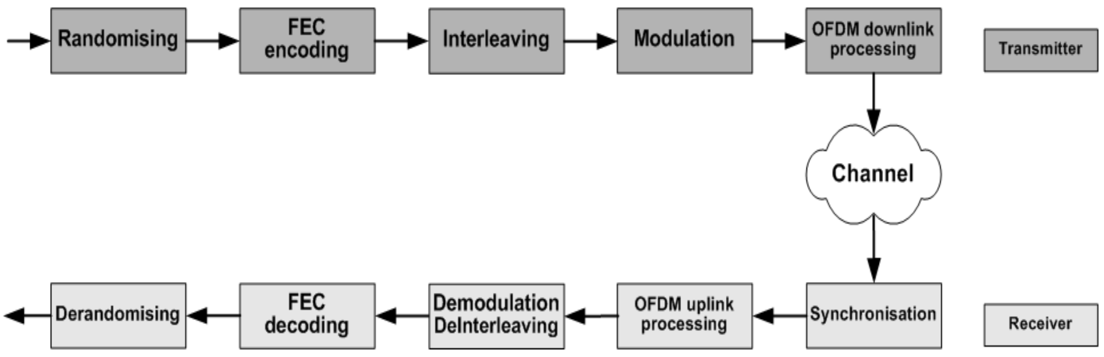


Figure 2.1 Fixed WiMAX physical layer processing chain

2.4.1 Randomising

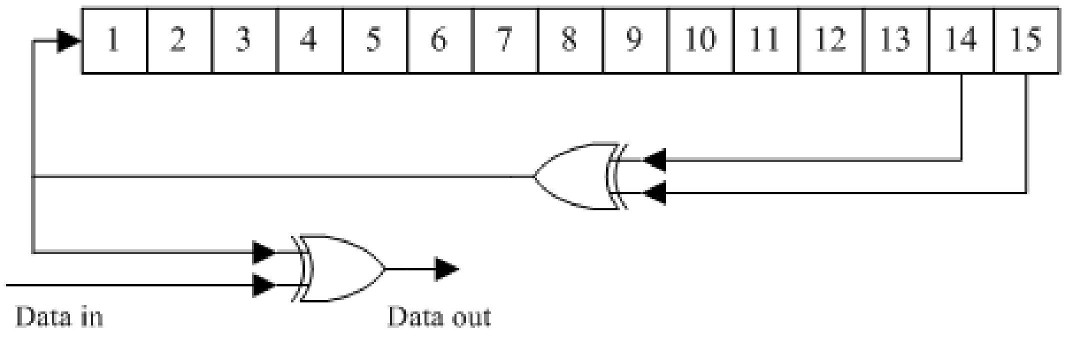


Figure 2.2 PRBS for the data randomisation

In WiMAX, randomisation is used to provide a simple encryption and prevent vicious receivers from decoding the data [2]. Each burst of information data is randomised in the transmitter side and de-randomised in the receiver side. Randomisation and de-randomisation are functionally equal by using a Pseudo-Random Binary Sequence (PRBS) with a generator polynomial of $X^{15} + X^{14} + 1$. The PRBS initialisation sequence is either a sequence of 100101010000000 or dependent on some parameters, such as OFDM symbol number and BS identification. The PRBS generator structure is shown in Figure 2.2.

2.4.2 FEC encoding

Forward Error Correction (FEC) coding is a channel coding, broadly used in wireless communication. FEC can detect and correct errors without the need to retransmit the data through the channel which may have a large propagation delay. It is achieved by incorporating redundant bits into the transmitted data and recovering the data in the receiver side. However, the consequence of this approach is a lower net bit rate caused by the introduced redundant bits [19]. In a WiMAX PHY, an FEC encoding is built by concatenating a Reed-Solomon (RS) block code and a punctured convolutional code [2]. The RS code is used as the outer code, and the convolutional code is used as the inner code. It means that the information data are firstly passed through the RS encoder in a block format and then encoded by a convolutional encoder followed by a data puncturing. A zero-tailing byte is appended to the end of each data package generated by RS encoding to reset the convolutional encoder state. In WiMAX, the RS encoding is applied with all modulation schemes except BPSK. Other FEC modes such as block and convolutional turbo codes are optional in the fixed WiMAX standard, and not addressed in this thesis.

2.4.2.1 Reed-Solomon encoding

Due to its outstanding performance for error correction especially for burst errors, Reed-Solomon encoding [20] is widely used in communication and storage systems. An RS (n, k) code can correct up to t errors, where n represents the data block size after encoding, k represents the data block size before encoding and t is equal to $(n-k)/2$. In WiMAX specifications, an RS (255, 239) code is used to correct up to 8 errors, each sized one byte. The code generator polynomial is shown as Equation 2.1 [2]

$$g(x) = (x+\lambda^0)(x+\lambda^1)(x+\lambda^2)\ldots(x+\lambda^{2t-1}) = \sum_{i=0}^{2t} g_i x^i \qquad (2.1)$$

where $\lambda = 0x02$. Both multiplication and addition operations in Equation 2.1 are performed in Galois Field (GF). Figure 2.3 shows the diagram of an RS encoder in WiMAX. Usually, the input block size $k^{'}$ is less than 239 bytes. Hence the first $239-k^{'}$ bytes need to be padded with zeros. As the RS (255, 239) code is systemic, the output block directly contains the input block. The information bytes are appended by 2T' bytes generated redundancy, where the value of T' depends on the modulation scheme and puncturing pattern.

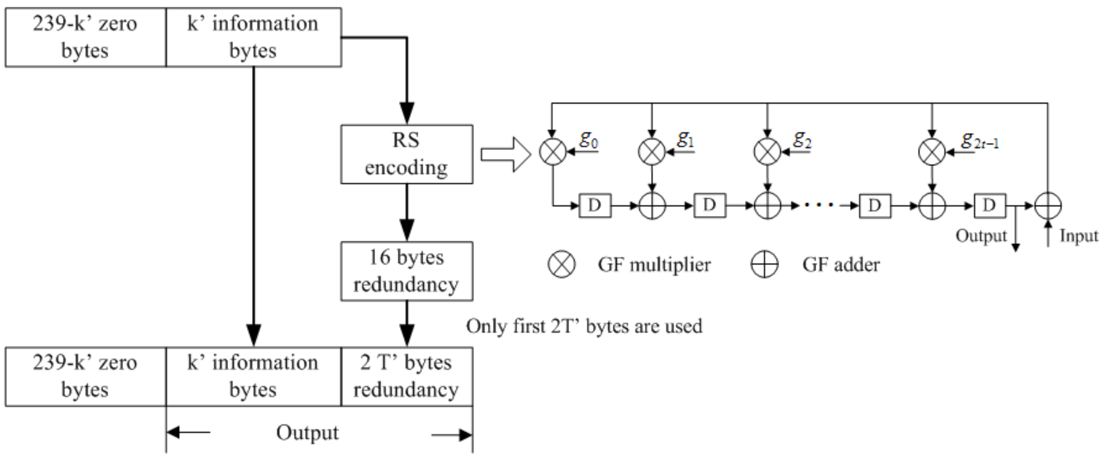


Figure 2.3 Reed-Solomon encoder

2.4.2.2 Convolutional encoding

A convolutional encoder is based on a linear finite-state shift register which consists of K m-bit registers and n modulo-2 adders, where K is called constraint length [21]. A convolutional code has n generator polynomials, each of which represents the connections of all registers to one modulo-2 adder. By passing each m-bit information through the encoder, n output bits are generated through n modulo-2 adders. The code rate is defined as $R = m/n$. Shown in Figure 2.4, WiMAX uses a binary convolutional encoder with a constraint length of 7 and a code rate of 1/2 [2]. For $m = 1$, only $K - 1$ one-bit registers are needed, since the input bit can be directly fed into the modulo-2 adders. The

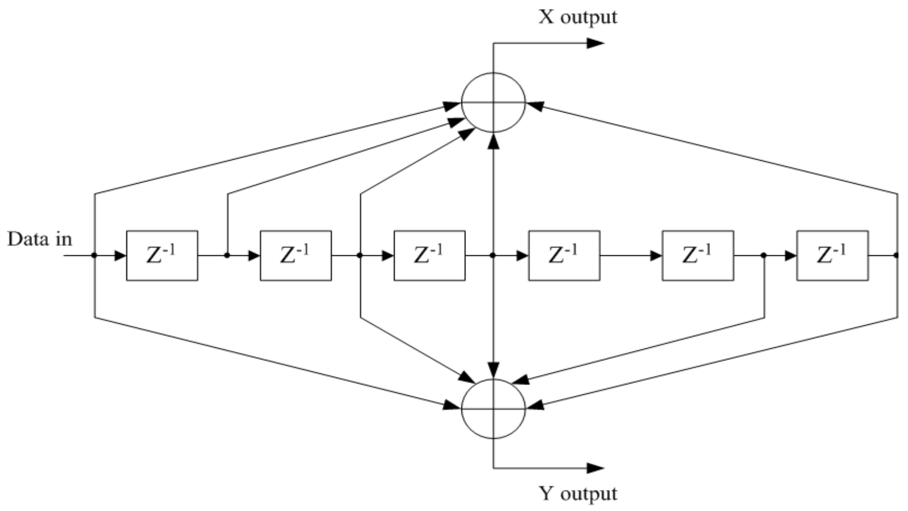


Figure 2.4 Binary convolutional encoder with a constraint length of 7 and a code rate of 1/2

two generator polynomials are G1(1,1,1,1,0,0,1) and G2(1,0,1,1,0,1,1), respectively.

In addition, theoretically the convolutional encoding operation can be graphically represented in three different ways: state diagram, tree diagram and trellis diagram. Among these diagrams, the trellis diagram is broadly used, due to its presentation of the passage of time. Here a brief introduction is given for the trellis diagram which is used in convolutional decoding as well. Figure 2.5 shows the trellis diagram for a simple convolutional encoder with a code rate of 1/2 and generator polynomials as G1(1,1,0) and G2(1,1,1). The horizontal axis represents time, while the vertical axis shows the encoder states which are the value of registers. Each new input bit causes a state transition which is represented by one connection from the current state to the next state through either a solid line for a 0 input or a dashed line for a 1 input. The corresponding output bits are shown in parentheses. The encoding operation starts from state 00. Figure 2.5 also highlights the encoding path for an input sequence of 1100 in red, where the most significant bit is fed into the encoder firstly. The generated output for this sequence is 11001001.

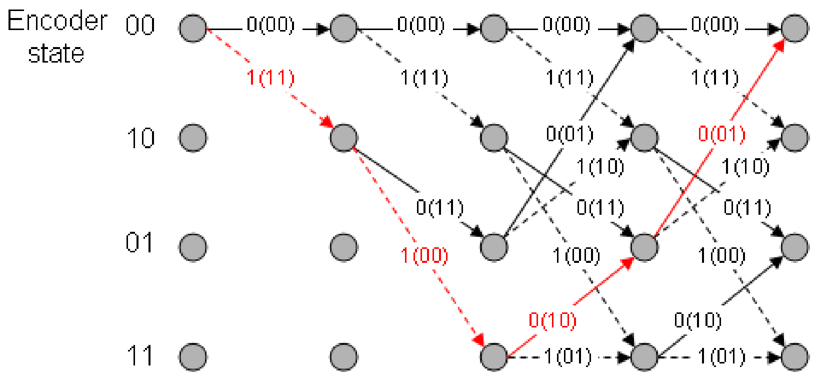


Figure 2.5 Trellis diagram representation of a simple convolutional encoder

2.4.2.3 Puncturing

The convolutional code in WiMAX has a native code rate of 1/2 which means for every input bit, the convolutional encoder will introduce one redundant bit. If a higher code rate is demanded, a different encoder is needed. However, data puncturing can be used after a convolutional code to achieve higher code rates and also improve the flexibility of the encoder without involving different encoders. During data puncturing, some bits in the encoder output are selectively deleted. Table 2.5 shows the puncturing patterns supported in WiMAX, which give different final convolutional code rates from 1/2 (no puncturing) up to 5/6 (minimal redundancy) [2]. In this table, “1” means corresponding bit is kept, otherwise removed.

Table 2.5 Puncturing patterns in WiMAX

Code rate	1/2	2/3	3/4	5/6
X	1	10	101	10101
Y	1	11	110	11010
Output	X_1Y_1	$X_1Y_1Y_2$	$X_1Y_1Y_2X_3$	$X_1Y_1Y_2X_3Y_4X_5$

2.4.3 Interleaving

The data transmitted in channels with multipath fading may suffer burst errors which can not be easily corrected by FEC codes. An effective method is interleaving the coded bits before the modulation, so that the burst errors can be distributed to different modulated data symbols and then corrected by FEC decoding [21]. In WiMAX, the interleaver involves two permutations given in Equations 2.2 and 2.3 [2], respectively.

$$m_k = (N_{cbps}/12)\times k_{\mathrm{mod}12} + floor(k/12) \qquad k = 0,1,\ldots,N_{cbps}-1 \tag{2.2}$$

$$j_k = s\times floor(m_k/s) + (m_k + N_{cbps} - floor(12\times m_k/N_{cbps}))_{\mathrm{mod}(s)} \quad k = 0,1,\ldots,N_{cbps}-1 \tag{2.3}$$

where k, m_k and j_k are the indices before and after the first and second permutations, respectively, and $s = ceil(N_{cpc}/2)$. N_{cpc} represents the number of coded bits per subcarrier and depends on the modulation scheme. While N_{cbps} represents the interleaver block size or coded bits per OFDM symbol, decided by both the modulation scheme and the number of subchannels. In the receiver side, de-interleaving is performed in two permutations as well, but in an inverse direction, given in Equations 2.4 and 2.5.

$$m_j = s \times floor(j/s) + (j + floor(12 \times j / N_{cbps}))_{\mathrm{mod}(s)} \qquad j = 0,1,\ldots,N_{cbps}-1 \qquad (2.4)$$

$$k_j = 12 \times m_j - (N_{cbps}-1) \times floor(12 \times m_j / N_{cbps}) \qquad j = 0,1,\ldots,N_{cbps}-1 \qquad (2.5)$$

where j, m_j and k_j are the indices before and after the first and second permutations, respectively.

2.4.4 Data modulation

After interleaving, the data bits are mapped to constellation points, which are complex valued symbols, by data modulation or symbol mapping. Each symbol can represent one-bit or several bit data according to the modulation scheme. In WiMAX, various modulation schemes are supported, including BPSK, Quadrature Phase-Shift Keying (QPSK), 16QAM

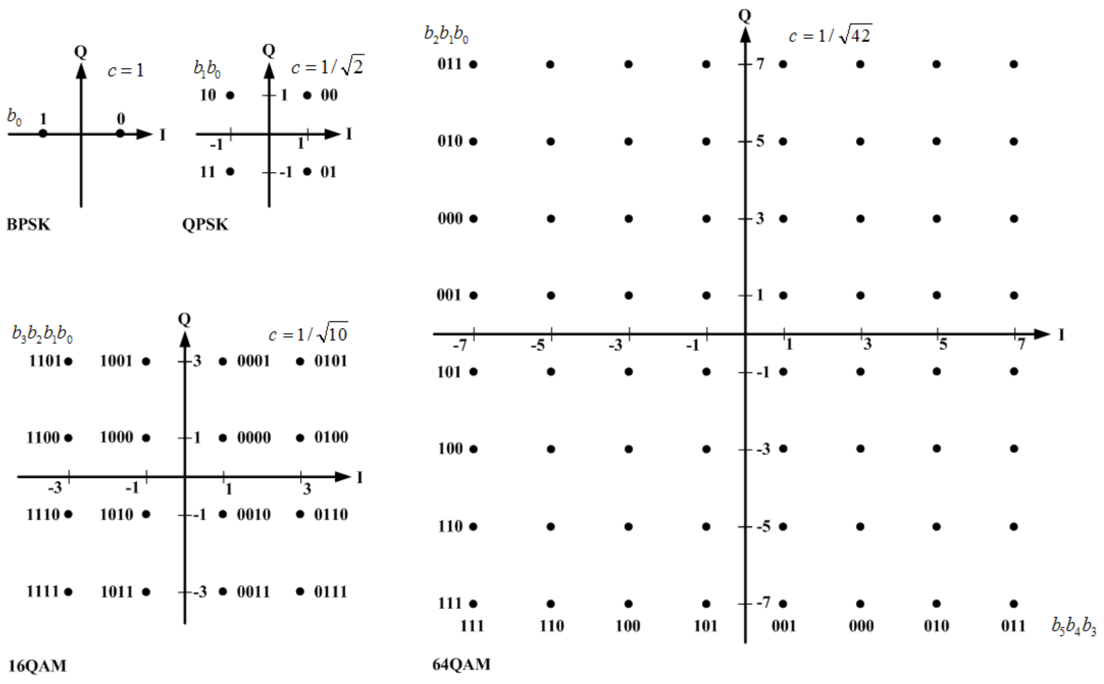


Figure 2.6 BPSK, QPSK, 16QAM and 64QAM modulation schemes in WiMAX

and 64QAM [2]. Among these schemes, BPSK is most robust and 64QAM offers the highest data rate. Figure 2.6 shows the constellations for each modulation scheme. For normalisation, the absolute value of each constellation point is multiplied with a factor c shown in Figure 2.6.

2.4.5 OFDM processing

One of the major challenges for high-data-rate wireless broadband systems is the Intersymbol Interference (ISI) due to the multipath propagation. ISI becomes very severe when the symbol time T_s is much less than the channel delay spread τ. It is especially true for WiMAX which has a long range and a high data rate [18]. In WiMAX, OFDM is used to overcome the ISI. OFDM is a popular multicarrier modulation technology widely used in many high-data-rate systems such as Wi-Fi and DSL. In a WiMAX physical layer, OFDM processing is one of key computation functions, composed of pilot insertion, DC and guard band insertion, inverse FFT (IFFT) and Cyclic Prefix (CP) extension in downlink processing together with CP removal, FFT, pilot removal, DC and guard band removal in uplink processing. In WiMAX, after data modulation, OFDM splits a high-rate single carrier into N low-rate subcarriers, each of which is orthogonal to each other and does not affect the encoded data on other subcarriers. Then OFDM uses IFFT to create a composite OFDM symbol stream by taking the encoded data of each subcarrier [22]. Therefore, the OFDM symbol time becomes $T_s * N$ which can be significantly larger than τ. Additionally through the use of CP, the channel can be made entirely ISI-free. Usually, for fixed WiMAX OFDM, there are 256 subcarriers consisting of 192 information subcarriers, 8 pilots, one Direct Current (DC) subcarrier and 55 guard subcarriers [2]. Figure 2.7 shows the frequency domain representation of an OFDM symbol for fixed WiMAX. While Figure 2.8 illustrates the OFDM downlink processing in the transmitter side for fixed WiMAX. In the receiver side, the OFDM uplink processing restores the high-rate single carrier by using FFT and removing CP, DC, guard band as well as pilots. In the following subsections, each sub-block

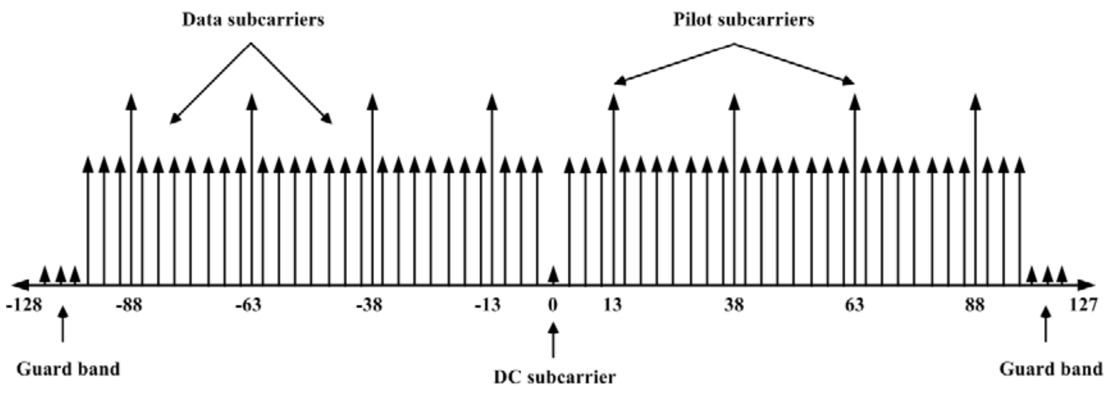


Figure 2.7 Frequency domain representation of an OFDM symbol in fixed WiMAX

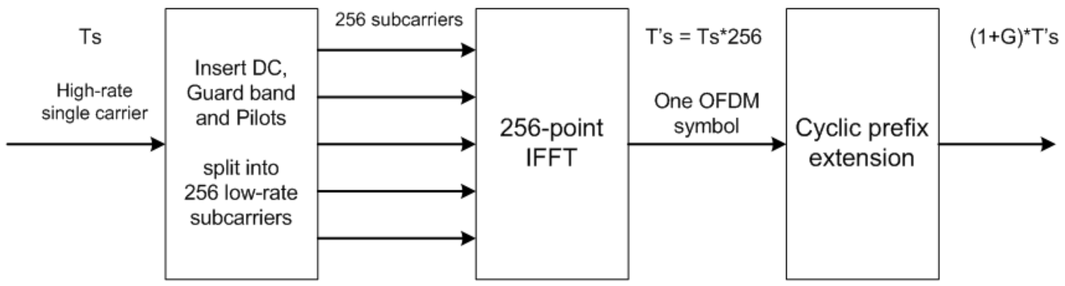


Figure 2.8 OFDM downlink processing in the transmitter side of fixed WiMAX

of OFDM is described.

2.4.5.1 Pilot modulation and insertion

In fixed WiMAX OFDM, eight subcarriers carry pilot signals which are used for synchronisation in the receiver side. The values of DL and UL pilots for OFDM symbol k are obtained from Equations 2.6 and 2.7, respectively [2], where w_k is generated by a PRBS with a generator polynomial of $X^{11} + X^{9} + 1$, as shown in Figure 2.9. The PRBS initialisation sequences for downlink and uplink are given in Figure 2.9 as well. Each pilot is BPSK modulated and inserted into the corresponding position of an OFDM symbol as illustrated in Figure 2.7.

DL: $$c_{-88} = c_{-38} = c_{63} = c_{88} = 1 - 2w_k \quad \text{and} \quad c_{-63} = c_{-13} = c_{13} = c_{38} = 1 - 2\overline{w_k} \qquad (2.6)$$

UL: $$c_{-88} = c_{-38} = c_{13} = c_{38} = c_{63} = c_{88} = 1 - 2w_k \quad \text{and} \quad c_{-63} = c_{-13} = 1 - 2\overline{w_k} \qquad (2.7)$$

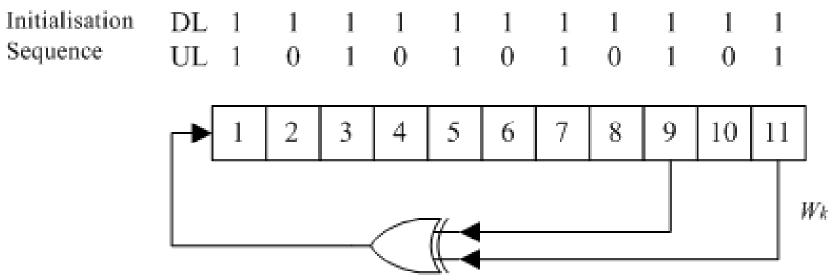


Figure 2.9 PRBS for pilot modulation

2.4.5.2 DC and guard band insertion

In fixed WiMAX OFDM, there are 56 null subcarriers which are not allocated any transmit power [2]. These subcarriers include one DC subcarrier and 55 guard subcarriers whose positions in OFDM frequency domain are shown in Figure 2.7. The DC subcarrier, whose frequency is equal to the centre frequency of radio frequency, is used to prevent any saturation effect or DC offset suffered by the receiver side. Guard subcarriers are located at the edge of the spectrum to reduce the interference between a WiMAX channel and its neighbour channels. Another reason to insert the guard band is that the size of FFT is always equal to $2n$, some dummy subcarriers should be padded to both edges [18].

2.4.5.3 FFT/IFFT

In OFDM processing, IFFT and FFT are used in the transmitter and receiver, respectively. An N-point IFFT transforms N frequency domain data into N time domain data, while FFT performs the opposite operation. IFFT and FFT have very similar structures. There are various algorithms to implement FFT/IFFT including radix-2 [23], radix-4 [24], split-radix [25] and so on. In this thesis, a simple and popular radix-2 algorithm, which is suitable for $2n$ point FFTs, is used. Figure 2.10 shows the data flow graph of an 8-point radix-2 FFT, where the horizontal axes signify the computation stages. There are in total 3 stages for an 8-point radix-2 FFT, and each stage has four butterfly operations. A signal flow

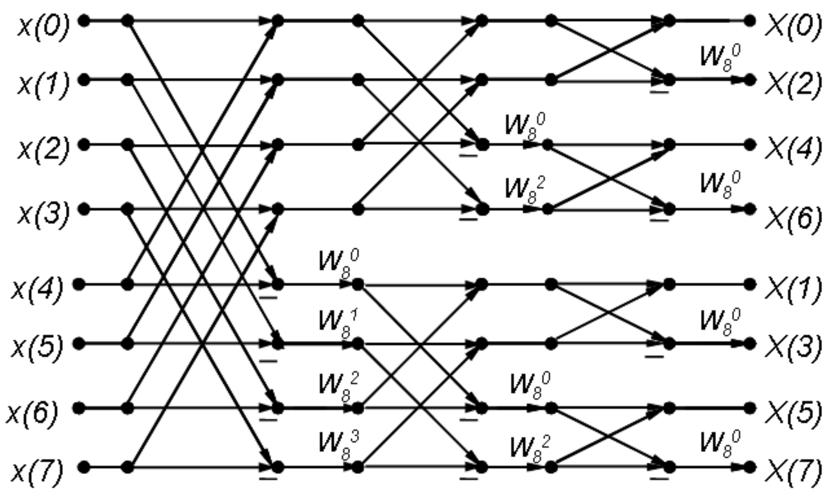


Figure 2.10 Data flow graph of an 8-point radix-2 FFT

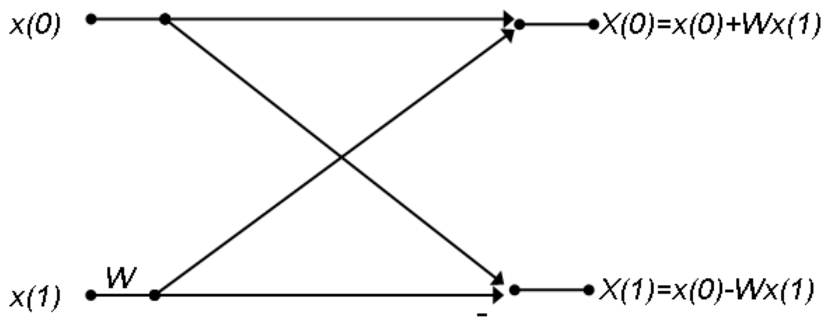


Figure 2.11 Signal flow representation of a radix-2 butterfly

representation of a radix-2 butterfly is given in Figure 2.11 where W, called twiddle factor, is a trigonometric constant coefficient. After the transformation, the order of output data should be reversed into the numerical order.

2.4.5.4 Cyclic prefix extension

After the IFFT operation, the interference between symbols is eliminated. However each OFDM symbol still suffers Inter-Carrier Interference (ICI) or ISI within one OFDM symbol. It is because of the feature of the multicarrier modulation where many subcarriers are tightly packed into one channel, so that even small frequency shifts will cause ICI [22]. To overcome ICI, a guard time is inserted in front of each OFDM symbol. In OFDM, the guard time is implemented by duplicating a tail portion of an OFDM symbol at the beginning of this symbol. This method is termed as cyclic prefix extension. In the receiver side, the CP will be removed before FFT performs. The ratio of CP length to useful symbol length is called guard fraction G. As shown in Figure 2.8, after the cyclic prefix extension, the symbol time becomes $(1+G)\times T'_s$. Obviously, a long guard time or large G will reduce the net bit rate, while a too short guard time or small G would not protect symbols from ICI. In WiMAX, G can have a value of {1/4, 1/8, 1/16, 1/32}. Many multi-core WiMAX solutions (e.g. Freescale WiMAX solution [26]) use 1/16 as the value of G. For the sake of comparison, a fraction of 1/16 is used in this thesis.

2.4.6 Synchronisation

The synchronisation block performs the symbol timing synchronisation to determine the symbol arrival time offset and the optimal timing instant by using the maximum likelihood (ML) estimation algorithm [27]. As shown in Figure 2.12, assume that we observe 2*N*+*L* consecutive samples represented in a 1-D vector r $[r(1)\cdots r(2N+L)]$ and these samples contain one complete CP extended OFDM symbol consisting of *N*+*L* samples, In Figure 2.12, *N* is the length of an OFDM symbol, L is the length of CP, and θ is the unknown time offset. The ML estimation of θ can be obtained by Equation 2.8 [28].

$$\overset{\wedge}{\theta}_{ML} = \arg \underset{\theta}{Max}\{|\gamma(\theta)| - \rho\Phi(\theta)\} \tag{2.8}$$

where $\gamma(\theta) = \sum_{k=\theta}^{\theta+L-1} r(k)r^*(k+N)$, $\Phi(\theta) = \frac{1}{2}\sum_{k=\theta}^{\theta+L-1} |r(k)|^2 + |r(k+N)|^2$ and $\rho = \frac{SNR}{SNR+1}$.

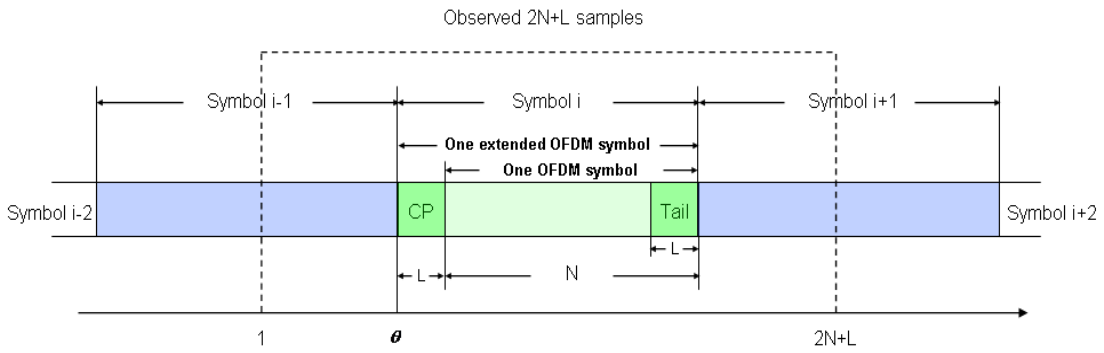


Figure 2.12 Timing synchronisation

2.4.7 Data demodulation

In the receiver side, the soft decision based data demodulation restores the original bits from the complex valued symbols by comparing the values of real and imaginary parts with some threshold values [2]. Figure 2.13 shows the soft decision demodulations for 16QAM where *d* is the threshold. 64QAM has more threshold values but maintains the similar idea of 16QAM demodulation. As for BPSK and QPSK, the original bits can be easily recovered according to the sign of the real and imaginary parts, as shown in Figure 2.6.

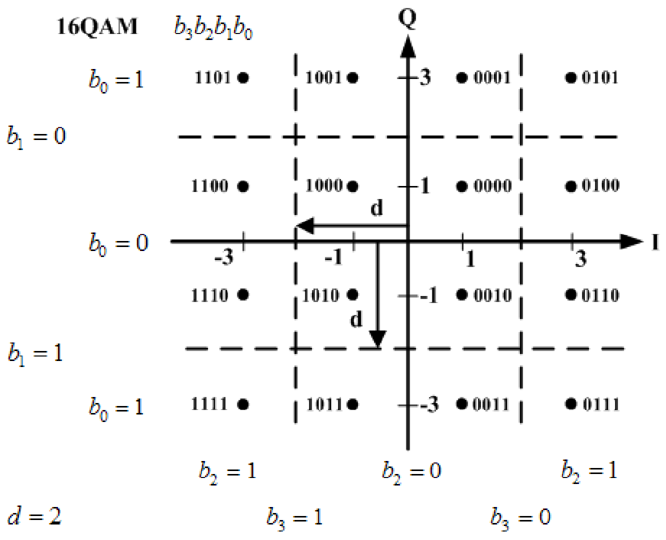


Figure 2.13 16QAM soft decision data demodulation

2.4.8 FEC decoding

In the receiver side, FEC decoding performs the inverse order of FEC encoding, and consists of de-puncturing, Viterbi decoding used to decode convolutional codes and Reed-Solomon decoding. The de-puncturing is simply performed by inserting 0 at the punctured positions. Viterbi and RS decoding are described in the following subsections.

2.4.8.1 Viterbi decoding

The basic idea behind the decoding of convolutional codes is to compare the received sequence with all possible valid sequences, by using the trellis diagram. In WiMAX PHY,

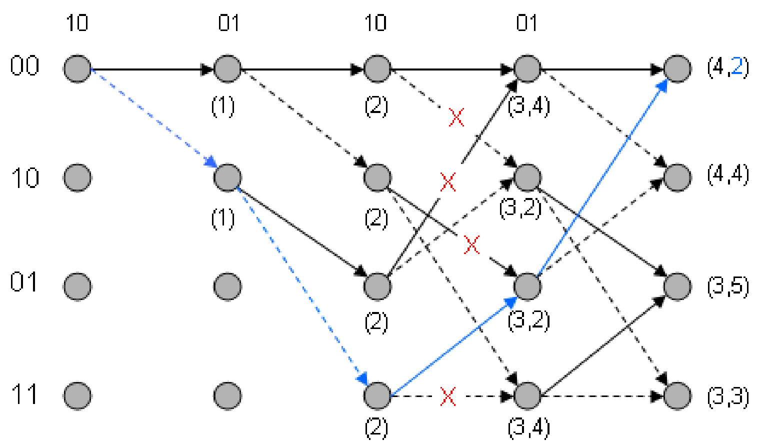


Figure 2.14 Viterbi decoding process for a convolutional code with a 1/2 code rate

the Viterbi algorithm [29] is applied to decode convolutional codes. The Viterbi algorithm allows multiple possible paths calculated concurrently and is much faster than other decoding algorithms like Fano algorithm which deals with only one path at a time. However, the Viterbi algorithm costs more resources if implemented in hardware. The Viterbi algorithm calculates branch metrics between the incoming bits and each possible branch of a state in the trellis diagram, and then accumulates the metric values along each path to generate a path metric. When two possible paths converge on one state node, the path with the lower path metric is kept as a survivor, with the other one discarded. Eventually, only the path with the lowest path metric is the ultimate survivor. Usually the Hamming distance is used as the metric. Figure 2.14 shows the decoding process for the convolutional code in Figure 2.5 by using the Viterbi algorithm. Assuming the received sequence is 10011001, it has two bit errors compared to the output sequence 11001001 of the convolutional encoder. In Figure 2.14, the Hamming distance for each path is shown in parentheses at each state point which the path reaches. The discarded paths have a red cross on them. Finally, the path with lowest Hamming distance of 2 is the survivor, highlighted in blue. The corresponding decoded sequence is 11001001, matching the output sequence of the encoder shown Figure 2.5.

2.4.8.2 Reed-Solomon decoding

In RS decoding, up to t errors (in WiMAX, t=8) can be detected and corrected following

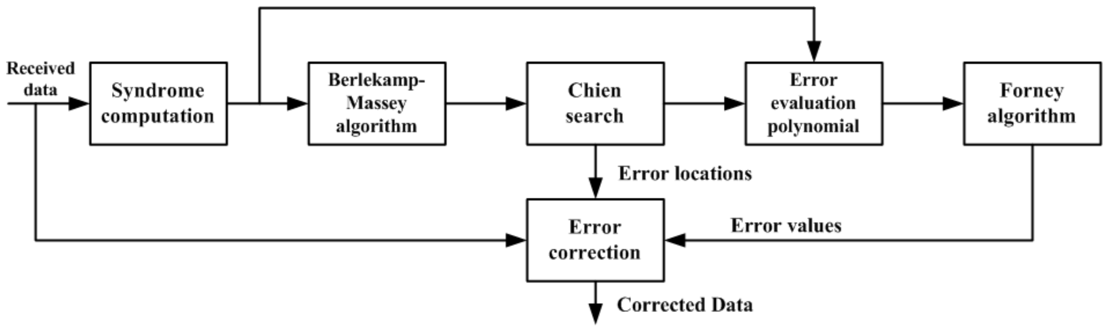


Figure 2.15 Reed-Solomon decoding process

the process shown in Figure 2.15 [30]. Firstly the received data is fed into the syndrome computation block for detecting whether the data contains any errors. Then the generated syndromes are used for retrieving the locations and values of errors. The Berlekamp-Massey algorithm [30] is chosen to generate an error-location polynomial. If the degree of the location polynomial is greater than t which is the maximum number of correctable errors, the decoder is only able to detect the presence of errors and will stop without any further correction. Otherwise the roots of this polynomial are obtained by an exhaustive search performed in Chien search block. Once the errors have been located, the error evaluation polynomial and Forney algorithm blocks will calculate the error values. Then error values together with error locations and received data are sent to the error correction block for recovering the original data. Each of blocks shown in Figure 2.15 involves plenty of GF calculations, the detail of which can be found in [30].

2.5 Summary

This chapter has reviewed current mainstream broadband access solutions including both wired and wireless technologies. Some comparisons were given to exhibit the advantages of WiMAX over its counterparts. Due to its merits, WiMAX was demonstrated to be a promising solution to promote ubiquitous Internet access. As the main target of this thesis is fixed WiMAX, Section 2.4 introduced the structure of fixed WiMAX physical layer and described the function and algorithm for each main block of both transmitter and receiver.

Chapter 3
Multi-core Processors and Reconfigurable Architectures

3.1 Introduction

Applications such as WiMAX demand high performance, strict low power, and in-field reprogrammability in order to follow evolving standards. Traditional single-core architectures could not satisfy all these requirements, since in the past few years, people have not seen great gains, but instead diminishing returns in single-core processor performance through increasing operating frequency. It is well known that the development of single-core processors hits three walls: memory wall, ILP wall and power wall. Now both industry and academia are all agreed that the continuing increases in the transistor count and operating frequency can no longer make a microprocessor faster, instead multiple or many simpler processing cores should be put on a chip. As a result, the processing load of complicated applications can be distributed across multiple processing cores. This approach can deliver more overall performance through parallelism, as well as consume less power.

On the other hand, in recent years, several new coarse-grained dynamically reconfigurable architectures have emerged, such as Transport Triggered Architecture (TTA) [31], Pleiades [32], DAPDNA-2 [33] and Reconfigurable Instruction Cell Array [7]. Incorporating a DSP-like flexibility as well as a close performance and power efficiency of ASICs [7], coarse-grained DR processors (e.g. RICA) are promising to be candidates for processing cores in high performance embedded multi-core processors.

Through gradually introducing the background knowledge of multi-core processors, reconfigurable architectures and RICA technology, this chapter indicates the reasons why RICA reconfigurable technology based multi-core architecture is explored in this thesis. This chapter is structured as follows. Section 3.2 investigates the advantages of multi-core processors over single-core processors and the taxonomies of multi-core processors. Section 3.3 introduces reconfigurable architectures in terms of different classifications. Section 3.4 focuses on a detailed introduction of a course-grained DR architecture – RICA, which is the target processing core used to build the proposed multi-core architecture in this thesis. Section 3.5 reviews the traditional WiMAX implementations on ASICs, GPPs and Field Program Gate Arrays (FPGAs). Section 3.6 addresses some existing work on multi-core WiMAX implementations. Section 3.7 summarises the reason why the RICA technology is chosen for the proposed multi-core architectures in this thesis.

3.2 Multi-core processors

As a kind of parallel architecture, a multi-core processor architecture combines two or more independent cores (e.g. GPP and DSP cores) into a single integrated circuit die. Actually parallel computing is not a new concept. Ever since the first effort to design a parallel machine (called SOLOMON) was made in 1958 [34], the parallel computing paradigm has been improving dramatically. Within the past half century, many advanced technologies have been created and become popular, such as Single Instruction Multiple Data (SIMD), Symmetric Multiprocessing (SMP) and multithreading. Now multi-core processors adopt many of these technologies which were originally developed for mainframes and supercomputers.

Currently many multi-core processors have been fabricated for different markets such as servers, personal computers and embedded systems. These multi-core processors include Intel Quad-Core Xeon [35], Nvidia GeForce GTX260 [36], ARM11 MPCore [37], Ambric Am2000 family [38] and so on. With the release of more and more multi-core processor products from top semiconductor companies, multi-core architectures are believed by many people as the trend of the development of processors and highly anticipated to exceed the common interpretation of Moore's Law [39].

3.2.1 The advantages of multi-core processors

Multi-core processors have many advantages. First of all, the salient characteristics of a multi-core processor enable it to avoid or move the three walls that single-core processors face further away. The placement of multiple processing cores on a single chip allows high-speed shared caches or stream buffers incorporated into the chip. The integrated shared memory blocks can operate at a much higher clock rates and have lower access latencies than on-board shared memory blocks which signals have to travel off-chip to access. Hence, the speed gap between processing cores and memory can be efficiently alleviated. Meanwhile thread level parallelism or multithreading has become more popular and can take better advantage of computing resources than ILP. In a multi-core environment, multithreading can bring potential into full play, since every core can support multithreading. Additionally, as multi-core processors can do more tasks or threads in parallel, multi-core processors can be designed to operate at lower frequencies than single-core processors. Theoretically, power consumption increases proportionally with the frequency, especially dynamic power consumption which is responsible for 80% of the overall power consumption [40]. Therefore multi-core processors can have a relaxed power budget, and mitigate the pressure in the cooling system. Moreover, if a processing core is not used, it can be switched off, so that the overall static power consumption can be reduced as well.

Secondly from the software point of view, multi-core processors can dramatically slash the response time for computation-intensive tasks like antivirus scan and video playback. It is achieved by assigning each such application to one core, so that the tasks will not compete for the computing resources on the same core as super-threading does on single-core processors. This design reuse has far less design risk than developing a new processor starting from scratch. In addition, multi-core processors can be fault-tolerant. If one core fails, the faulty core can be switched off. The rest of the system can still correctly function, even though delivering a reduced performance. Furthermore, a multi-core processor has advantages over a multi-chip module, which incorporates multiple die in one package, across all performance metrics. The advantages are mainly due to the shorter distances, which the signals between processing cores have to travel, as well as the reduced space multi-core processors require.

3.2.2 Taxonomies of multi-core processors

Multi-core processors can be categorised in term of different taxonomies such as the target markets, heterogeneity or homogeneity as well as the types of cores used to build the multi-core architectures.

3.2.2.1 Computing markets

Recording computing markets, multi-core processors are broadly used in three different markets, servers, desktop and laptop computing as well as embedded systems. These markets are distinct from each other in applications and requirements [41]. For servers, it is crucial to reliably and efficiently provide services such as Internet searching and online transactions. Commercial multi-core processors targeting servers include Intel Xeon [35], AMD multi-core Opteron [42], IBM dual-core POWER6 [43], Sun Microsystems eight-core UltraSPARCT1/T2 [44] and so on. The desktop and laptop computing market is continuously driven to improve price/performance ratio. This field is dominated by Intel and AMD x86 architecture based products. One exception is IBM dual-core PowerPC 970MP

[45] for Apple Power Mac G5 computer. Currently desktop and laptop multi-core processors contain up to four processing cores. Such processors include Intel Core 2 Quad [46] and AMD quad-core Phenom [47].

Recently with more and more computation-intensive applications, such as wireless communication and multimedia, incorporated into embedded systems, there is an increasing trend to use multi-core processors in embedded devices such as game consoles and WiMAX base stations. Many embedded systems demand real-time performance meaning that a service or an application must be completed before a deadline. Meanwhile portable embedded devices need to be highly energy efficient for extending battery life. Usually multi-core processors in this field are especially designed for some dedicated applications or application domains. Examples include the Cell processor [48] initially designed for PlayStation 3, Ambric Am2000 family [38] targeting video processing and picoChip multi-core DSPs [49] looking at wireless communication. Considering the issue of power consumption, a number of embedded multi-core processors are composed of many simpler processing cores with lower frequency, compared to their counterparts in the other two markets. For example, Am2045 [38] consists of 360 32-bit processors with a 333 MHz clock, while picoChip PC101 [50] contains 430 16-bit processors and runs at 160MHz.

3.2.2.2 Heterogeneity and Homogeneity

Depending on whether processing cores are identical, multi-core processors can be categorised into homogeneous multi-core and heterogeneous multi-core. As its name implies, all processing cores in a homogeneous multi-core architecture are same. A heterogeneous multi-core architecture consists of different cores, each of which could be optimised for certain applications running on it. It allows heterogeneous multi-core architectures offer better performance for domain specific applications. However, heterogeneous multi-core architectures may face more challenges than its homogeneous twin in terms of Application Programming Interface (API) support, task partitioning and mapping, as well as compiler and language development [51]. Now it is still debated that homogeneity or heterogeneity will be the future of multi-core processors [52]. Recently most multi-core processors used in servers as well as desktop and laptop computing are homogeneous. As to embedded systems, both homogeneity and heterogeneity architectures are widely used. The rest of this section describes more details about heterogeneous multi-core processors, since homogeneous ones are built by replicating processing cores.

Each core of a heterogeneous multi-core architecture may differ in size, functionality, performance and complexity or even come from totally different processor families. Heterogeneous multi-core architectures can be further divided into single Instruction Set Architecture (ISA) heterogeneity and multi-ISA heterogeneity. Multi-ISA heterogeneity architectures contain processing cores based on different ISAs and hence may involve multiple tool flows. The hybrid feature makes this type of architecture more difficult to design and program for. In contrast, all cores in a single-ISA heterogeneous multi-core architecture share a same ISA. It allows any core to execute any application used to run on other cores, with little effort on system reconfiguration and program modification. Previously mentioned commercial products, such as the Cell processor and picoChip multi-core DSPs, are heterogeneous.

A variety of heterogeneous multi-core systems have been proposed in academia as well, based on either single-ISA [53-55] or multi-ISA [56]. The authors in [53] proposed a single-ISA heterogeneous multiprocessor consisting of two processor types EV5 and EV6. It demonstrated that heterogeneous architectures could provide significantly higher performance than their equivalent-area homogeneous counterparts. In [54], a heterogeneous multiprocessor architecture was proposed for high definition video. Using Silicon Hive technology, this architecture employed five processors based on three different hardware templates. In [55], the authors presented a heuristic to efficiently explore the design space for a pipeline based heterogeneous multiprocessor system. This system uses a commercial Application-Specific Instruction-Set Processor (ASIP) – Xtensa LX from Tensilica [57].

JPEG and MP3 applications were taken as case studies. Shikano et al. [56] presented a heterogeneous CMP consisting of three SuperH processors and two FE-GA processors for a MP3 encoder.

3.2.2.3 Core types

Multi-core processors can be structured with varied types of processing cores such as GPP cores, DSP cores, Graphics Processing Unit (GPU) cores, ASIP cores, dynamically reconfigurable cores or a mix of these. Most multi-core processors used in servers as well as the desktop and laptop computing are based on GPP cores, except those used for special purposes such as Nvidia GeForce GTX200 GPUs. Due to the variety of embedded systems, there are many more choices in building embedded multi-core processors. For instance,

Table 3.1 Comparison of multi-core processors

Multi-core	No. of Cores	Target market	Homo/Heterogeneity	Core type
Opteron [42]	2/4/6	Server	Homogeneous	GPP
POWER6 [43]	2	Server	Homogeneous	GPP
Processor in [53]	8	Server	Heterogeneous	GPP
UltraSPARC T1/T2 [44]	8	Server	Homogeneous	GPP
Xeon [35]	2/4/6	Server	Homogeneous	GPP
Core 2 Duo/Quad [46, 58]	2/4	Desktop/laptop	Homogeneous	GPP
GeForce GTX200 [36]	10	Server/desktop/laptop	Homogeneous	GPU
Phenom [47]	3/4	Desktop	Homogeneous	GPP
PowerPC 970MP [45]	2	Desktop	Homogeneous	GPP
Am2045 [38]	336	Embedded	Homogeneous	DSP
ARM11 MPCore [37]	2/3/4	Embedded	Homogeneous	GPP
Cell [48]	9	Embedded	Heterogeneous	Mix
DAPDNA-2 [33]	2	Embedded	Heterogeneous	Mix
IXP 2350 [59]	5	Embedded	Heterogeneous	Mix
MSC8126 [26]	4	Embedded	Homogeneous	DSP
PC101/102 [49, 50]	430/308	Embedded	Heterogeneous	DSP
MRC6011 [60]	6	Embedded	Homogeneous	DR
Processor in [55]	4/5/6/9	Embedded	Heterogeneous	ASIP
Processor in [56]	5	Embedded	Heterogeneous	Mix
SB3010 [61]	5	Embedded	Heterogeneous	Mix

Freescale MSC8126 multi-core DSP [26] is equipped with four SC140 VLIW DSP cores, ARM11 MPCore [37] can be configured to have up to four GPP cores, and Freescale MRC6011 [60] consists of six coarse-grained DR cores. Moreover, some embedded multi-core processors are built with a mix of different types of cores. Examples of such multi-core processors include Intel IXP 2350 [59] and IPFlex DNPDNA-2 [33]. IXP 2350 is a network processor which has an XScale GPP core and four microengine cores, designed for line-rate packet processing. DNPDNA-2 is comprised of a high-performance Reduced Instruction Set Computer (RISC) core called DAP and one coarse-grained reconfigurable fabric called DNA. Obviously, such kind of multi-core processor must be heterogeneous, but heterogeneous multi-core processors are not necessarily to be built with different types of cores. Table 3.1 summarises some popular commercial and academic multi-core processors in terms of the number and types of cores and their targets.

3.3 Reconfigurable architectures

Reconfiguration Architectures (RAs) refer to systems containing a Reconfigurable Processing Unit (RPU) which can be customised to execute different applications after fabrication through multiple configuration bits. Basically, an RPU is an array of computational elements connected through a set of programmable routing resources [62]. Offering a good balance between implementation efficiency and flexibility, reconfigurable architectures can achieve much higher performance than GPPs and DSPs, while maintaining software-like flexibility by post-fabrication programmability. To date, there is no unified taxonomy established for reconfigurable architectures. Usually several architectural characteristics are used to classify RAs, including execution structures, granularity and reconfiguration schemes [63].

3.3.1 Execution structures

According to the structure of executing datapaths and control paths, reconfigurable architectures can be classified into two categories [63]. In the first category, a RA is a combination of a RPU for datapath operations and a main processor for control operations. Actually, the RPU acts as a hardware accelerator to execute the computation-intensive part of the application, while the main processor (e.g. RISC or VLIW based) is usually responsible for control operations, RPU configuration and the synchronisation with the RPU. This kind of architecture may involve plenty of manual work on separate programming for the RPU and the main processor as well as the synchronisation between the two different components. The typical examples of such architectures are Montium [64] and MorphoSys [65]. According to the type of coupling of the RPU to the main processor, this category can be further split into loose coupling, where the RPU connects to the processor through internal or external bus, and tight coupling where the RPU is even embedded into the processor and communicates with it through shared registers [66]. In the second category, the RPU can execute both datapath and control operations. Although a processor may exist in this kind of RA, there is no need for moving a large mount of data and manual synchronisation between the processor and the RPU. Such architectures include Pleiades [32] and RICA [7].

3.3.2 Granularity

The granularity represents the size of smallest function units in a reconfigurable architecture. Recording this parameter, reconfigurable architectures can be categorised into fine-grained, coarse-grained, medium-grained, very coarse-grained and mixed-grained RAs [66].

3.3.2.1 Fine-grained reconfigurable architectures

Fine-grained RAs contain basic function units sized down to one bit. Many early FPGAs and Programmable Logic Devices (PLDs) mainly consist of fine-grained basic logic blocks such as transistors, NAND gates, an interconnection of multiplexers and Look-Up Tables (LUTs) [67]. Among these fine-grained logic blocks, LUTs are broadly used in commercial FPGAs/PLDs such as Xilinx Virtex 5 [68] and Altera Stratix III [69]. Typically, a LUT is implemented by using Static Random Access Memory (SRAM) cells and can be used to build any bit-level logic function with the same number of inputs [67]. Based on bit-level blocks, fine-grained RAs can be configured as a wide range of hardware circuitries. The flexibility comes at a price of significant routing overheads and a large number of configuration bits, since each basic function unit should be connected and configured. As a result, dedicated applications can not efficiently perform on fine-grained RAs, and consume more power, space and configuration time with fine-grained implementations [66]. For example, assuming an implementation of a 32-bit adder on a fine-grained FPGA, it will cost logic blocks and corresponding routing resources. Obviously, this implementation is inefficient for addition-intensive applications, compared to a reconfigurable architecture with an array of 32-bit adders.

3.3.2.2 Coarse-grained reconfigurable architectures

Usually coarse-grained RAs are based on word-level building blocks which could be Arithmetic Logic Units (ALUs) or some custom function units (e.g. adders and multipliers) connected through interconnection networks such as mesh, crossbar and linear array [70]. Made less generic and more specific to some application domains, coarse-grained RAs have several advantages over their fine-grained counterparts in terms of power consumption, speed and area costs. However, some possible inefficient implementation on coarse-grained RAs could happen when operations required by applications do not match the size of building blocks. For example, an application requiring 8-bit addition operations could waste resources of an RA built with only 32-bit blocks. Actually this kind of situation can be avoid by either using vector operations or tailoring the resources to meet the requirement since the application could be known in advance.

In the last twenty years, many commercial and academic coarse-grained RAs have been proposed. In Table 3.2, a summary is given for the comparison of various coarse-grained RAs. As shown in Table 3.2, most listed coarse-grained RAs are based on a format of a

Table 3.2 Comparison of coarse-grained reconfigurable architectures

Architecture	Execution structure	Programmability	Target application
Adapt2400 [71]	RPU for control/datapath	SiliverC (ANSI C derivative)	Signal/image processing
ADRES [72]	VLIW for control; RPU for datapath/limited control	ANSI C	Video
DAPDNA-2 [33]	RISC for control; RPU for datapath	MATLAB/Simulink; ANSI C	General purpose
HiveFlex CSP [73]	VLIW for control; RPU for datapath	ANSI C	OFDM
Matrix [74]	RISC for control; RPU for datapath	Assembly level macro language	General purpose
Montium [64]	RPU as a co-processor for datapath	Montium LLL language (low level)	Wireless, image/ signal processing
MorphoSys [65]	RISC for control; RPU for parallel-data operations	Assembly	Pixel-processing
PACT XPP [75]	RPU as a co-processor for datapath	NML language (low level)	DSP
PADDI-2 [76]	RPU as a co-processor for DSP datapath	Assembly	DSP
PipeRench [77]	RPU as an accelerator for pipeline applications	DIL single-assignment language (C subset)	Pipelining
Pleiades [32]	Main processor for control; RPU for datapath/control	Netlist for RPU	Multimedia
RaPiD [78]	RISC for control; RPU for datapath	RaPiD-C	Pipelining
RICA [7]	RPU for control/datapath	ANSI C	Wireless, image and signal processing
TTA [31]	RPU for control/datapath	ANSI C	DCT, Viterbi and etc.

control-responsible main processor paired with a computation-responsible RPU acting as a coprocessor or a hardware accelerator. In Section 3.3.1, it has been pointed out that these architectures will involve separate programming and complicated synchronisation. It is also shown that many of coarse-grained RAs are programmed in low level languages (e.g. assembly and netlist) or C subsets and extensions which are not fully compatible with ANSI C. In Section 3.4, a newly emerging coarse-grained reconfigurable architecture RICA is described in details. This architecture supports ANSI C programming and is able to run both control path and datapath operations.

3.3.2.3 Medium-grained and very coarse-grained reconfigurable architectures

Medium-grained structures have granularities sized between fine grains and course grains. Examples of such architectures are Garp [79] and CHESS [80]. Garp [79] has an array of 2-bit reconfigurable ALUs, while the kernel of CHESS architecture [80] is a hexagonal array of 4-bit ALUs targeting multimedia applications. As for very coarse-grained architectures, the basic reconfigurable building blocks are based on a processor. In [81], RAW is built with 16 tiles, each of which is a 32-bit modified MIPS R2000 microprocessor. While REMAEC [82] has an 8 by 8 array of 16-bit nanoprocessors.

3.3.2.4 Mixed-grained reconfigurable architectures

Currently many modern commercial FPGAs are built with a mixture of different logic function units from fine-grained blocks like LUTs to coarse-grained blocks, such as dedicated memory block, multipliers and adders [67]. These FPGAs are heterogeneous structures and called mixed-grained RAs. Basically such a FPGA (e.g. a Xilinx or Altera FPGA) contains an array of Configurable Logic Blocks (CLBs) [68] or Adaptive Logic Modules (ALMs) [69] and routing resources. CLBs or ALMs are connected through a reconfigurable interconnection network. Usually a CLB or an ALM contains several 4-input or 6-input LUTs, flip-flops, multiplexers and even some arithmetic units. Meanwhile this kind of FPGA feature an amount of block RAMs for meeting different memory configurations and word-level arithmetic blocks like multipliers for accelerating computation-intensive applications. These coarse-grained blocks have grown to take dominant space on mixed-grained FPGAs. For example, the latest Xilinx Virtex 6 FPGAs contain up to 864 DSP48E1 slices and 1064 dual-port RAM blocks. Each DSP48E1 slice consists of a 25 x 18 multiplier, an adder, and an accumulator. Each block RAM can store 36 Kbits [83]. Some dedicated functions (e.g. DSP functions) can be effectively implemented on these coarse-grained blocks and thus have significant advantages in terms of area, speed and energy, compared to their implementations on CLBs or ALMs. However, if these functions are not present in the target applications, those special blocks will become redundant and thus reduce the area efficiency of mixed-grained FPGAs [67].

3.3.3 Reconfiguration schemes

In terms of the reconfiguration scheme, RAs can be classified into static, dynamic and partially dynamic reconfiguration [66]. In the static reconfiguration, the hardware is configured at system power up. Once the operation starts, the hardware will remain the same configuration through the whole life of the application. One example of such statically reconfigurable devices is Altera Stratix II FPGAs [84]. In contrast of the static reconfiguration, dynamically reconfigurable architectures can be run-time configured to execute different configuration contexts (the configurations for applications are partitioned into multiple contexts) by swapping or switching these contexts. Most coarse-grained architectures are dynamically reconfigurable. Both static and dynamic reconfigurations need the devices to be fully reconfigured. Rather than this full reconfiguration, some devices such as Xilinx Virtex 5 FPGAs [68] allow to selectively reconfigure a part of resources, in the meantime keeping the rest of resources operating. This partial dynamic reconfiguration can hide the reconfiguration time in the computation time.

3.4 Reconfigurable instruction cell array

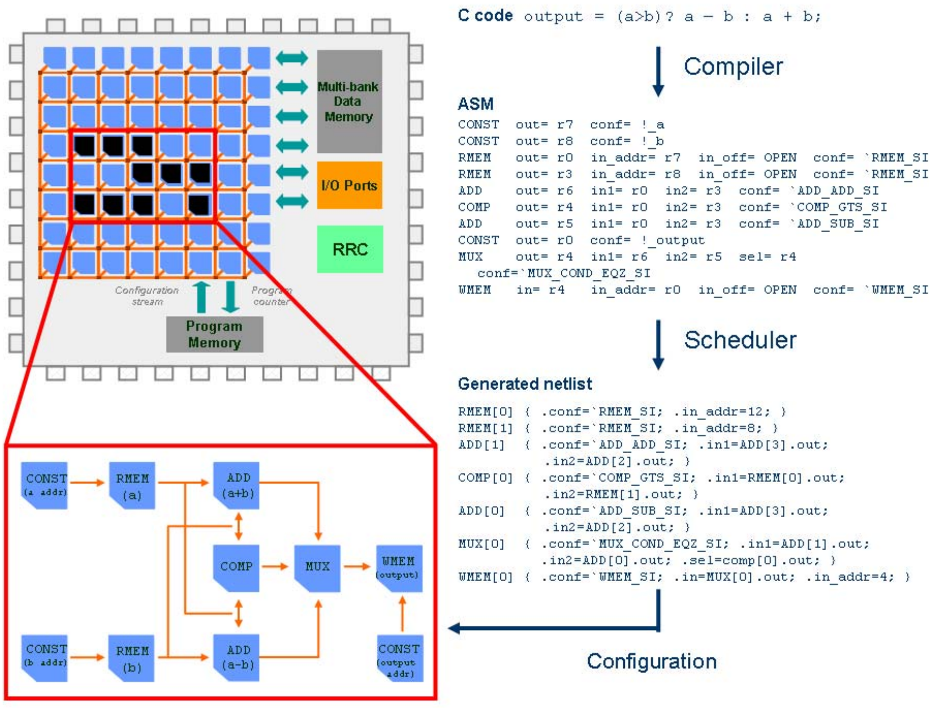
In [7], a coarse-grained dynamically reconfigurable architecture is proposed, called reconfigurable instruction cell array. In this thesis, the proposed multi-core architecture uses processing cores based on this RICA architecture. The following subsections describe the RICA architectural characteristics and its tool flow.

3.4.1 Architecture

The main part of the RICA architecture is a heterogeneous array of instruction cells interconnected through an island-style mesh fabric. Each instruction cell can be configured to do a small number of operations as listed in Table 3.3 which gives the description of both standard instruction cells and some existing custom instruction cells. Figure 3.1 shows the architecture view of RICA. One salient characteristic of RICA is that the array can be customised at the design stage according to application requirements in terms of the number of each certain cell, and even allows for new custom cells added. Another characteristic is that RICA supports operation chaining – the ability to execute both dependent and independent instructions in parallel in one configuration context (called step), which leads to

Table 3.3 RICA Instruction Cells

Standard Instruction Cell	Associated Operations
ADD	Addition and subtraction
MUL	Multiplication
REG	Registers
CONST	Interconnection
SHIFT	Shifting
LOGIC	Logic operation (e.g. XOR and AND)
COMP	Comparison
MUX	Multiplexing
I/O REG	Register with access to external I/O ports
RMEM	Interface for reading data memory
WMEM	Interface for writing data memory
DMA_interface	Interface for DMA
I/O port	Interface for external I/O ports
RRC	Controlling reconfiguration rates
JUMP	Branches
Custom instruction Cell	**Associated Operations**
SOURCE	Interface for reading files
SINK	Interface for writing files
GFMULT	Galois Finite Field Multiplication
SBUF	Interface for accessing stream buffer banks
MULTIPTBK_REG_FILE	Interface for accessing shared register files



C code `output = (a>b)? a – b : a + b;`

ASM

```
CONST  out= r7  conf= !_a
CONST  out= r8  conf= !_b
RMEM   out= r0  in_addr= r7  in_off= OPEN  conf= `RMEM_SI
RMEM   out= r3  in_addr= r8  in_off= OPEN  conf= `RMEM_SI
ADD    out= r6  in1= r0  in2= r3  conf= `ADD_ADD_SI
COMP   out= r4  in1= r0  in2= r3  conf= `COMP_GTS_SI
ADD    out= r5  in1= r0  in2= r3  conf= `ADD_SUB_SI
CONST  out= r0  conf= !_output
MUX    out= r4  in1= r6  in2= r5  sel= r4
  conf=`MUX_COND_EQZ_SI
WMEM   in= r4   in_addr= r0  in_off= OPEN  conf= `WMEM_SI
```

Generated netlist

```
RMEM[0] { .conf=`RMEM_SI; .in_addr=12; }
RMEM[1] { .conf=`RMEM_SI; .in_addr=8; }
ADD[1]  { .conf=`ADD_ADD_SI; .in1=ADD[3].out;
          .in2=ADD[2].out; }
COMP[0] { .conf=`COMP_GTS_SI; .in1=RMEM[0].out;
          .in2=RMEM[1].out; }
ADD[0]  { .conf=`ADD_SUB_SI; .in1=ADD[3].out;
          .in2=ADD[2].out; }
MUX[0]  { .conf=`MUX_COND_EQZ_SI; .in1=ADD[1].out;
         .in2=ADD[0].out; .sel=comp[0].out; }
WMEM[0] { .conf=`WMEM_SI; .in=MUX[0].out; .in_addr=4; }
```

Figure 3.1 The RICA architecture

high degrees of parallelism.

In contract of traditional processors which have computation units on critical paths pipelined to improve the throughput, the RICA architecture introduces variable clock cycles to ensure longer critical paths consume more clock cycles. An instruction cell termed Reconfiguration Rate Controller (RRC) is used to achieve that. RRC is a counter-like cell which contains the amount of clock periods (called RRC periods) needed for the critical path of the current step.

Once counting down to zero, RRC generates an *Enable* signal for the program counter and registers. The RRC period can be programmable as part of the RICA's configuration [7]. Secondly, the distinction from a conventional processor is RICA's memory access patterns (for both data and program memory). The RICA architecture provides multiple memory interface cells which allow for simultaneously reading and writing multiple data memory locations within one single step. Meanwhile, the instruction stream fetch patterns are unusual in that successive iterations of certain loops can be executed following only a single fetch from the program memory, if the loop can be placed into one single step. This kind of step is called a kernel. In addition, the instruction words are also quite large compared to those in existing DSPs. Thirdly a multi-bank memory system is provided to meet the higher data memory bandwidth demanded by the operation chaining in order to keep the array fed with data. This is quite different to that seen in conventional processors as well.

As a reconfigurable architecture, RICA needs to be reconfigured for each updated step. The time consumed by loading the next step and configuring the array is called configuration latency. The configuration latency is variable according to the number of instruction cells used and configured in each step. Basically, the execution time of a step is greater than the configuration latency. Therefore, the configuration latency between consecutive steps can be hidden through prefetching the next step when the current step is executed. The prefecthing can work only if the current step has no branches or has an unconditional branch. The RICA architecture does not need to be reconfigured when the step loops to itself, in which case there is no configuration latency at all [7].

3.4.2 Standard tool flow

RICA supports the development from high level languages such as C in a manner very similar to conventional microprocessors and DSPs. Figure 3.2 illustrates the standard tool flow of RICA. As shown in Figures 3.1 and 3.2, ANSI C programs can be compiled into a sequence of steps captured in a netlist, by means of a compiler and a scheduler specific to the RICA architecture. These steps are switched between several times during the execution of the complete program. Formed by a sophisticated scheduling algorithm [85], each step packs together as many chains of operations as possible, and takes a variable number of clock cycles to execute, in order for all chains to complete. The contents of each step are executed concurrently by RICA according to the availability of hardware resources.

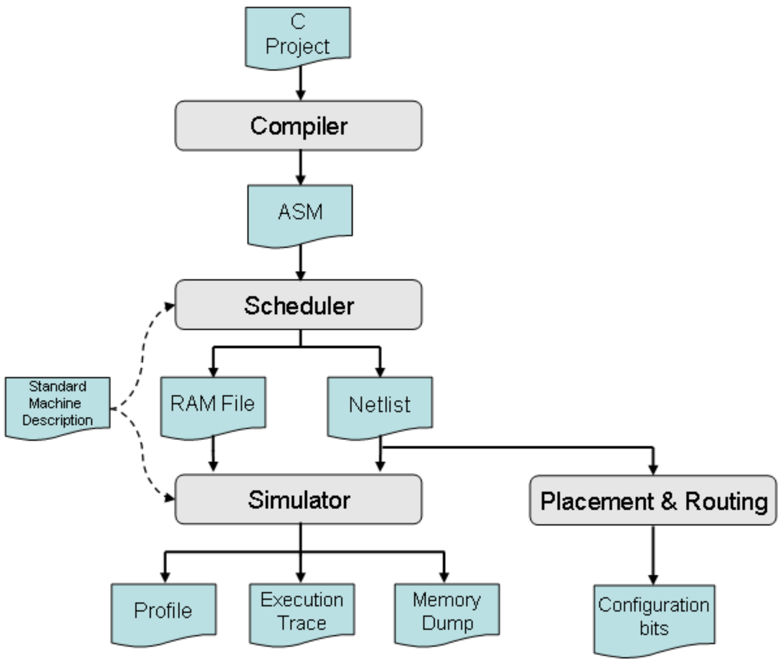


Figure 3.2 Standard RICA tool flow

The generated steps can be executed on a SystemC based simulator or converted into configuration bits by the placement and routing tool. The generated configuration bits can be downloaded into the program memory on a real RICA chip. As shown in Figure 3.2, a Machine Description File (MDF) containing the architecture and cell information is used as an input to both the scheduler and the simulator. A number of algorithms have been tested to demonstrate that RICA can achieve up to 8 times higher throughput than RISC processors such as OR32 [7].

3.5 Traditional WiMAX silicon implementations

Traditionally, there are various WiMAX implementation solutions including custom chips, GPP/DSP based System on Chips (SoCs), FPGAs and so on. These solutions are still popular to some extent, even owning some non-neglectable drawbacks. These solutions are described in the following subsections.

3.5.1 Custom chip implementations

Custom chip implementations are traditionally popular in designing wireless communication protocols. Usually those chips are fully customised for WiMAX applications and designed through the standard ASIC design flow involving Register Transfer Level (RTL) coding, logic synthesis and layout design. One such example is Intel WiMAX Connection 2400 chip [86] which can deliver a maximum sustainable throughput at 20Mbps for DL and 5Mbps for UL. Another is Wavesat NP7256 [87] providing a throughput up to 37.5Mbps. Other commercial custom chip solutions include TeleCIS Wireless TCW 1620 chip, Philips UXF234xx series and so on. Obviously, these ASIC based implementations can offer high throughput, high power efficiency and small footprints for WiMAX applications. However as they are fully customised, this kind of solution is inherently inflexible and can not be upgraded and altered after fabrication. This is one of its main drawbacks, particularly when this approach is used for wireless communication applications like WiMAX where the standards always keep changing. Moreover, designing full custom chips requires more human effort and can not meet the short time-to-market demands.

3.5.2 GPP/DSP based SoC implementations

Instead of designing basic components from RTL, another popular solution for design companies is to use the third-party or their own GPP and DSP IPs to build SoCs for WiMAX. This kind of solution usually uses GPPs for Media Access Control (MAC) layer protocols and DSPs for PHY processing. One such example is Intel Pro/Wireless 5116 [88] which features dual-core ARM 946E-S engines for MAC, PHY and protocol processing as well as a DSP engine for OFDM processing. Some other SoC solutions, such as Fujitsu MB873400 and ST STW51000, have MAC protocols performed by GPPs, but PHY processed by their own proprietary ASICs. This kind of solution is more flexible and significantly reduces the product development cycle, compared to full custom chips. However, as Application Specific Standard Products (ASSPs), SoC solutions are still time-consuming and expensive. In conclusion, it is very costly to enter the WiMAX market with either the SoC solution or the custom chip solution.

3.5.3 FPGA/PLD implementations

WiMAX protocols can be mapped on FPGAs/PLDs for fast hardware prototypes and some domains where the costs of power and area are not important concerns. FPGA/PLD based solutions can provide more flexibility and shorter time-to-market compared with the above two types of solutions. Announced in [89], Altera Stratix II FPGAs can be used to address implementation challenges of designing a WiMAX system. Based on a 90-nm SRAM process, Stratix II FPGA family owns up to 18k logic elements, 9 Mbit on-chip Random Access Memory (RAM) and 384 18-bit multipliers. The authors in [90] implemented a fixed WiMAX baseband on an Altera Stratix EP1S80 board containing an Altera EPM7064 PLD which has more resources than normal FPGAs. About 57% of logic modules were occupied to perform this implementation. In [91], another FPGA solution was presented to implement a fixed WiMAX PHY on a Lattice ECP33 device. This implementation cost around 70% of resources on this device which has 33k LUTs, 8 coarse-grained sysDSP blocks. Both Altera and Lattice provide IPs to accelerate computation-intensive functions like RS coding, Viterbi decoding and FFT. In addition, some other researches developed incomplete WiMAX implementations on Xilinx FPGAs. For example, in [92], the authors presented a WiMAX transmitter implementation on a Xilinx Virtex II Pro FPGA, using a pure VHDL mapping approach and Xilinx AccelDSP tool, respectively. However, FPGAs/PLDs are not power and area efficient for WiMAX implementations, as they cost much more power consumption and area than ASICs [93].

3.6 Multi-core implementations of WiMAX

Recently, a significant demand for high performance and flexible WiMAX solutions is forcing designers to move to multi-core systems. Several multi-core solutions, targeting WiMAX applications, are introduced in the following subsections. These solutions are different from each other in terms of architectures and core types. For example, both Freescale MSC8126 DSP and picoChip PC102 are built with DSP cores, while others use mixed core types.

3.6.1 PicoArray

PicoArray PC102 [49], from picoChip, is a heterogeneous multi-core DSP based on a tiled architecture. It features 308 16-bit processors in three different variants as well as various hardware accelerators. Based on the Harvard architecture and a common instruction set, each processor variant has varying amounts of separate local data and program memory, and extra custom instructions for facilitating dedicated functions. The inter-processor communication and synchronisation is achieved through signals. All processors are bonded to picoBus which connects many programmable bus switches to build up a deterministic interconnect. The inter-processor communication is time division multiplexing based, and determined during the compilation time, so that the communication bandwidth is guaranteed. Moreover each picoArray has four inter-picoArray interfaces which allow it to connect to up to four other picoArrays for exchanging data and extending the system. Figure 3.3 shows the picoArray interconnection and one example signal path for the inter-processor communication. In the picoChip PC7218 baseband reference design [94], two PC102 chips are used for WiMAX PHY processing and can achieve a data rate up to 37Mbps, with PC8520 fixed WiMAX software reference design [95].

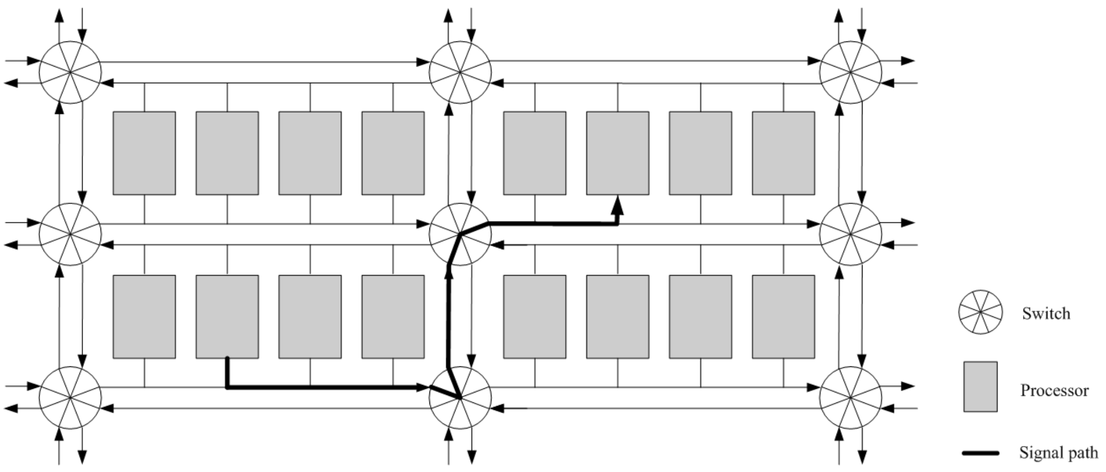


Figure 3.3 PicoArray interconnection and the inter-processor communication

3.6.2 Freescale MSC8126

Freescale's MSC8126 multi-core DSP [96] consists of four SC140 extended cores, a Viterbi coprocessor and a turbo coprocessor. Each extended core features an SC140 DSP core operating at 400 MHz or 500 MHz, 224 Kbyte level-1 data memory and 16-way 16 Kbyte instruction cache. The level-l memory of each core can be accessed by any other cores for fast data transfer. All the extended cores share an internal 475 Kbyte level-2 memory through a prioritised round-robin mode. In [26], an MSC8126 DSP handles a fixed WiMAX PHY processing, with one single core for the transmitter and the other three for the receiver. The allocated tasks and loading for each SC140 core are shown in Table 3.4.

Table 3.4 MSC8126 loading and task allocation

	Allocated tasks	Loading
Core 1	Randomising, FEC encoding, interleaving, modulation and OFDM processing in the transmitter side	81%
Core 2	Synchronisation and OFDM processing in the receiver side	77%
Core 3	Demodulation, de-interleaving and Viterbi decoding	70%
Core 4	RS decoding and de-randomising	52%

3.6.3 Cell Broadband Engine

Cell Broadband Engine [48] is jointly developed by IBM, Sony and Toshiba, and composed of one Power Processor Element (PPE) and eight Synergistic Processing Elements (SPEs). The PPE is based on the Power architecture and acts as a controller for SPEs by running Linux. The SPEs are RISC processors with 128 bit SIMD acceleration, and carry out the most of computational workload. The PPE and eight SPEs are linked together through a circular ring bus which contains four channels and can deliver totally twelve transactions concurrently. The Cell processor can execute 256 Giga Floating-Point Operations Per Second (GFLOPs), running at 3.2 GHz. The initial target of the Cell processor was multimedia acceleration and vector processing, but latter is used for many other applications such as scientific computation and wireless communication. In [97], the authors implemented WiMAX PHY on a Cell processor where two SPEs used for the transmitter and three used for the receiver, while the PPE handles the management and control. This WiMAX implementation can achieve 20Mbps for both transmitter and receiver.

3.6.4 Intel IXP 2350 network processor

Targeting broadband access applications such as WiMAX and DSL, Intel IXP2350 network processor [59] consists of four fully programmable multithreaded microengines and one Intel XScale core. The microengines and XScale core can run up to 900MHz and 1.2GHz, respectively. Announced in the product brief of Intel's NetStructure WiMAX Baseband Card [98], the Intel IXP2350 processor can provide a data rate up to 36 Mbps. In this implementation, the microengines are programmed to perform PHY processing and the MAC layer is mapped on the XScale core. In IXP2350, there are various technologies to ensure the low latency communication between microengines targeting PHY. For example, next neighbour registers can speedup data and information transfer between adjacent microengines. Reflector mode pathways ensure the performance of unidirectional buses for delivering data and global event signals between microengines. In addition, ring buffers allow the intermediate data propagated along the pipeline built with microengines.

3.6.5 Sandbridge SB3010

Sandbridge's SB3010 chip [99] contains an ARM9 RISC core and four multithreaded Sandblaster DSP cores connected by a deterministic ring network. Each DSP core supports up to 8 threads and runs at 600MHz. In [61], the authors proposed an ANSI C based software implementation of fixed WiMAX on an SB3010 chip. In this design, one DSP core is utilised to implement the transmitter, while the receiver is mapped to the other three cores. Among overall used 24 threads, IFFT function is replicated across three threads for operating three different OFDM symbols. A round-robin scheduling is used to manage the data communication for this replication partition. This implementation delivers a data rate of

2.9 Mbps based on BPSK modulation and 1/2 rate convolutional code.

3.7 RICA based multi-core architecture

As described in Section 3.2, varied types of processing cores or their mix can be used to build up multi-core architectures. As for the high performance embedded systems market (e.g. consumer electronics), coarse-grained reconfigurable processing cores are very promising candidates, due to their high flexibility, performance and power efficiency. Among numerous coarse-grained reconfigurable architectures, RICA has many advantages over others, including high level programming support (i.e. ANSI C), the ability to execute both control path and datapath operations, and the ability to execute both independent and dependent instructions in parallel in one configuration context. Therefore, the RICA technology is chosen to implement the basic processing core in multi-core architectures proposed in this thesis.

As mentioned in Section 3.4, the key component of the RICA architecture is an instruction cell array which can be customised for dedicated applications before manufacture. In this thesis, the research is carried out to explore multi-core RICA instead of a single-core RICA with a very big instruction cell array. It is not only due to the advantages of multi-core architectures over single-core architectures described in Section 3.2, but also because the current RICA architecture does not support multithreading. It means a bigger single-core RICA can not have high area efficiency and a high resource utilisation rate for each configuration context.

3.8 Summary

This chapter described the advantages of multi-core processors and investigated the categories of multi-core processors in terms of different taxonomies. In this thesis, a coarse-grained dynamically reconfigurable processor – RICA is chosen as the target processing core for the proposed multi-core architecture. Hence this chapter introduced the background knowledge of reconfigurable architecture and especially highlighted the architectural characteristics and tool flow of RICA. Meanwhile various traditional and multi-core WiMAX implementations were addressed.

Chapter 4
A RICA Processor based Multi-core Architecture

4.1 Introduction

The last chapter introduced the background knowledge of multi-core processors and reconfigurable architectures, especially a coarse-grained dynamically reconfigurable architecture - RICA. The RICA architecture can address all the desired requirements for high performance embedded systems [7]. Meanwhile its cell array can be tailored towards different application domains. This chapter presents a multi-core architecture using RICA processing cores. Chapters 6 and 7 will develop multi-core solutions for WiMAX, based on this proposed architecture.

Currently, there are a few multi-core processors based on coarse-grained DR processors, including Freescale MRC6011 [60] and the work in [100]. Freescale MRC6011 consists of six reconfigurable compute fabric cores connected through a flexible and high-speed fabric. Each of the cores features one optimised 32-bit RISC engine and a coarse-grained reconfigurable computing array. In each core, the RISC engine supports C programming and handles the configuration and management of the array. Therefore, it is unavoidable that this architecture involves significant synchronisation overheads between RISC engines and reconfigurable computing arrays. In [100], a scalable and modular multi-core architecture template is proposed based on Silicon Hive reconfigurable processing cores. Each core is comprised of an array of Processing and Storage Elements (PSEs) built around a control processor. Both the control processor and PSEs are programmed using standard C, however, the timing synchronisation between them has to be coded manually. Another example is the work on developing a multi-core processor based on Montium tile processor [64] from RECORE. However, currently Montium tile processor does not support development from high-level languages. Although Montium is application customisable, this processor is not efficient in terms of the hardware resource occupancy, built with five identical 16-bit ALUs.

This chapter is structured as follows. Section 4.2 presents the overall multi-core architecture. In this section, subsections 4.2.1 through 4.2.6 describe processing cores, the memory architecture and other main components, respectively. Among these subsections, Section 4.2.2 addresses various architectural choices of memory available in this architecture, including shared/local data memory, shared register file and stream buffer. The access to the later two options requires custom instruction cells. Section 4.3 investigates the synchronisation methods for inter-processor communications and atomic operations developed for supporting synchronisation. Section 4.4 introduces how to integrate a custom instruction cell into an RICA processor.

4.2 The proposed multi-core architecture

Typically, multiprocessors (a multi-core processor is a multiprocessor) can be classified into two types, master-slave multiprocessing and symmetric multiprocessing [101]. In a master-slave multiprocessing, there is one master processor and multiple slave processors. The master processor assigns and schedules tasks to slave processors, controls and manages interrupt system and I/O peripherals, and runs the operating system if the processor support operating systems. The advantage of master-slave multiprocessing is its simplicity, since only the master processor can manage tasks and access all resources. The Cell processor is an example of mater-slave multiprocessing [48]. However, the master processor becomes a bottleneck, because all slave processors have to access resources through the master processor. On the contrary, each processor in SMP has equal power, equal access to all resources and can schedule tasks itself. Therefore SMP is more efficient but much complex than master-slave multiprocessing [101]. SMP is broadly used in multi-core processors with a small number of cores such as desktops, laptops and servers.

In this thesis, a master-slave multi-core architecture is proposed as shown in Figure 4.1. Actually, from the hardware point of view, this architecture is SMP. All processing cores are based RICA technology. Each of them can access the shared memory and has its own interrupt controller. However, one of cores is used as the master core which takes charge of task management and will run operating system in future. Therefore this architecture has the SMP-like efficiency but maintains the software simplicity of master-slave multiprocessing. In the context of this thesis, this architecture is referred as a master-slave multi-core architecture. In addition, this architecture can be configured as Single Program Multiple Data (SPMD) mode where all processing cores execute a same task but with different data.

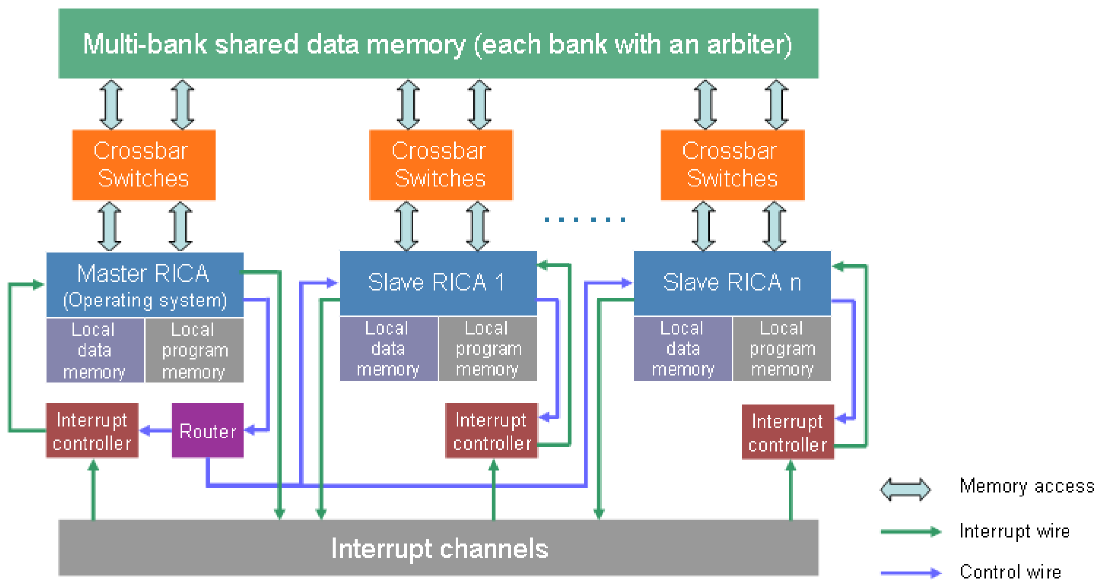


Figure 4.1 Proposed master-slave multi-core architecture

Described in Chapter 5, the MRPSIM simulator has a command line option --*arch* which can set the simulator to work as either master-slave mode or SPMD mode.

Consisting of one master core and multiple slave cores, this architecture is designed to be a basic architecture used in both homogeneous and heterogeneous multi-core WiMAX solutions introduced in Chapters 6 and 7. This architecture contains an interrupt system and a flexible memory architecture. One of the key characteristics of this architecture is that it provides support for DR processors (e.g. RICA) that may issue multiple concurrent memory access requests per cycle.

4.2.1 Processing cores

All processing cores including both the master and slaves in the multi-core architecture are based on the 32-bit RICA architecture, since the RICA architecture is suitable for executing both data path and control path programs. Each processing core has a set of memory addressed control and status registers, an RRC unit, a prefetching unit and multiple memory interface cells. Currently the main difference in functionality between the master and slaves is that the master core is used to take charge of the task management. When the system starts up, the master dispatches tasks to slaves by sending the task information through a router. The task information sent by the master is written to a few registers in a slave which then uses the information to choose a task to run. When a slave finishes its current task, it will send a request for a new task to the master through an interrupt. Then the master dispatches the information of a new task to that slave. Alternatively, as shown in Figure 4.1, an Operating System (OS) can be ported to the master core for dynamically scheduling and managing tasks. Except this control functionality, the master can do same computation workloads as slaves. In addition, the instruction cell array of each core can be tailored to particular tasks that the processing core is intended to execute.

4.2.2 Memory architecture

This multi-core architecture has a hierarchical memory architecture including shared memory, local memory, shared register file and stream buffer. Among these memory blocks, the shared register file is designed for loop level partitioning used for Viterbi and FFT applications (the details will be described in Chapter 6), and stream buffers are designed for image processing applications such as Freeman demosaicing. In the following subsections, these memory blocks are described in detail.

4.2.2.1 Shared memory/local memory

To address the parallel memory access requirement of RICA cores, this multi-core processor architecture is based on the Harvard architecture [41] where each core owns a program memory and all processing cores share a multi-bank data memory. Besides a shared data memory, each processing core can have its own local multi-bank data memory which cannot be addressed by other cores. Due to the adjacency to processing cores and small size, a local memory can have much lower access latency than the shared memory. By storing non-shared data in a local memory, the conflict in accessing the shared memory can be alleviated, and thus the throughput can be improved. In addition, each data memory bank has an arbiter which arbitrates memory accesses to the bank from different cores or different memory access interface cells of a same core. Table 4.1 describes the modes and

Table 4.1 Access modes and configuration bits for RMEM and WMEM cells

Mode		Configuration bits	Description
RMEM	RMEM_SI	001	Read a single word
	RMEM_SE_HI	010	Read a half word with sign extension
	RMEM_ZE_HI	011	Read a half word with zero extension
	RMEM_SE_QI	100	Read a quarter word with sign extension
	RMEM_ZE_QI	101	Read a quarter word with zero extension
WMEM	WMEM_SI	00	Write a single word
	WMEM_HI	01	Write a half word
	WMEM_QI	10	Write a quarter word

configuration bits of RICA memory access interface cells used to access both shared and local memory. Furthermore, for eliminating memory conflicts caused by simultaneous accesses to the shared data from different RICA cores, synchronisation methods (e.g. spinlock and semaphore) are developed to protect the shared data from multiple accesses at any given time. The detailed explanation is provided in Section 4.3.

4.2.2.2 Shared register file

Besides shared and local data memory introduced in the last subsection, this multi-core processor architecture also supports shared register files. Usually, a shared register file can operate at a much faster speed than normal memory blocks and be used to speed up data exchange in small scale as well as synchronisation between processing cores. Multi-port register files are commonly used in VLIW processors which need to access several registers simultaneously. It is well known that the number of ports and the size of the register file affect its energy consumption, access time and area [102]. Most of the previous work on the register file has been related to techniques of reducing the access time and power consumption.

In [103] the authors used techniques to split the global micro-architecture into distributed clusters with subsets of the register file and functional units. Similarly, the authors of [102] proposed the use of distributed schemes as opposed to a central implementation. Multi-level register file organisations have also been introduced to reduce the size of register files [104]. All the works mentioned above focus on reducing the number of registers. Other techniques, such as the one in [105], split the register file into interleaved banks, reduce the total number of ports in each bank, but retain the idea of a centralised architecture. In this thesis, a shared register file architecture is proposed. This architecture is split into independent banks with a reduced number of ports per bank. Each bank has one write port, one read port and 32x32-bit registers.

A custom register file interface cell, called MULTIPTBK_REG_FILE, is created and integrated into the RICA processor to support accessing distributed shared register files. The diagram of this interface cell is shown in Figure 4.2, and the pin description is given in Table 4.2. This cell supports various access modes controlled by configuration bits, shown in Table 4.3. Among these modes, MPREGFILE_LLNLY and MPREGFILE_SCNLY are used for atomic operations which are described in Section 4.3. The number of register file banks can be parameterised in the MDF. Figure 4.3 illustrates a design of a 4-bank shared register file where each bank has an individual arbiter. All instruction cells in RICA cores can be connected to shared register interface cells, and each interface cell is able to connect

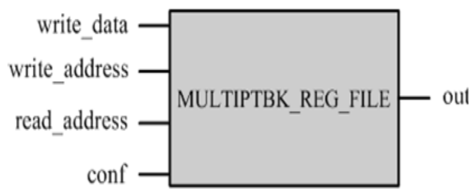


Figure 4.2 Custom instruction cell MULTIPTBK_REG_FILE

Table 4.2 Pin descriptions of MULTIPTBK_REG_FILE cells

Name	Type	Bit width	Description
write_data	In	32	Data written to register files
write_address	In	5	Write address
read_address	In	5	Read address
conf	In	3	Configuration bits
out	Out	32	Data read from register files

Table 4.3 Access modes and configuration bits of MULTIPTBK_REG_FILE cells

Mode	Configuration bits	Description
MPREGFILE_RDNLY	000	Read only
MPREGFILE_RDBFWRT	001	Read before write
MPREGFILE_RDAFWRT	010	Read after write
MPREGFILE_WRTNLY	011	Write only
MPREGFILE_LLNLY	100	Load link only
MPREGFILE_SCNLY	101	Store conditional only

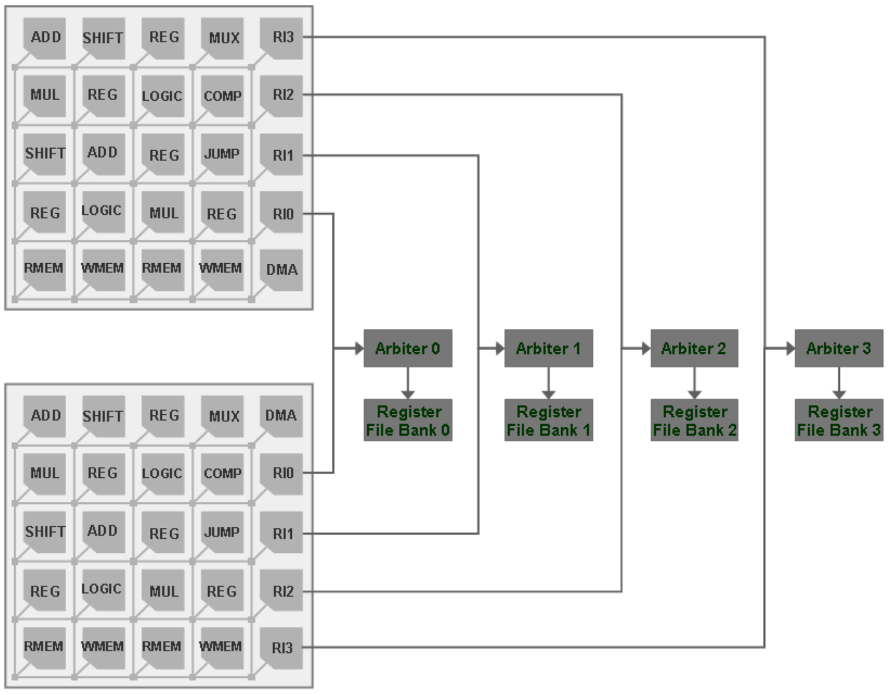


Figure 4.3 A 4-bank shared register file in the proposed multi-core architecture

to only one register file bank. For example, register file interface cell RI0 is only connected to Bank0, while interface cell RI1 can only access Bank1. This scheme keeps all possible connections between these interface cells and the shared register banks with a reduced interconnection complexity. The detail of how to introduce a new custom cell for a RICA processor is given in Section 4.4.

4.2.2.3 Stream buffer

Stream buffers are widely used in many processors to facilitate data transfer for image, video and communication applications, such as in Storm-1 stream processor [106]. In this proposed multi-core architecture, stream buffers are implemented as bidirectional bridges between each two adjacent processing cores for a mass data exchange, dramatically reducing the shared memory bandwidth requirement. The stream buffers can only be accessed by directly connected processing cores through a custom interface cell called SBUF which is provided by the extended RICA processor. All other instruction cells in RICA cores can be connected to these SBUF cells. Each SBUF cell can be used to access any stream buffer near to the processing core.

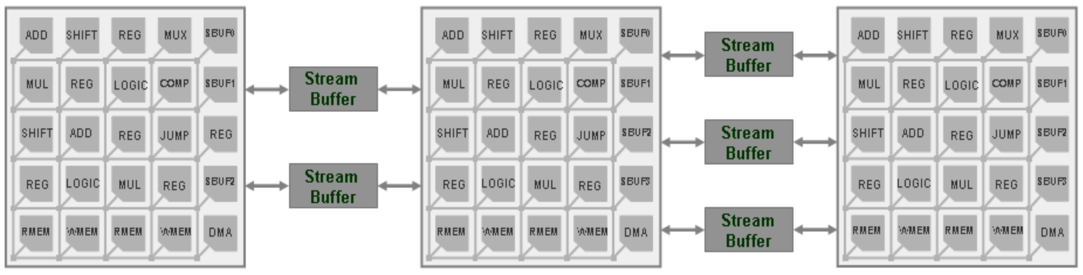


Figure 4.4 Stream buffers in the proposed multi-core architecture

Figure 4.4 illustrates an example which has three cores and a total of five stream buffers. As shown in Figure 4.4, the stream buffers are mainly used to store the intermediate data between processing cores in a pipelined processing mode where the first core (the leftmost one) reads input data from the memory block. The processing cores can set the initial writing or reading address for connected buffers at a random access mode, then push data to or pop data from the buffers. After each push or pop, the address will be automatically increased. The pin description and access modes of SBUF cells are given in Tables 4.4 and 4.5,

Table 4.4 Pin description of SBUF instruction cells

Name	Type	Bit width	Description
write_data	In	32	Data written to stream buffers
write_address	In	12	Initial write address
read_address	In	12	Initial read address
conf	In	3	Configuration bits
out	Out	32	Data read from stream buffers

Table 4.5 Access modes and configuration bits of SBUF cells

Mode	Configuration bits	Description
SBUF_SET_READ	101	Set initial read address
SBUF_SET_WRITE	110	Set initial write address
SBUF_SET_READ_WRITE	111	Set initial read and write address simultaneously
SBUF_STREAM_READ	001	Read stream buffer
SBUF_STREAM_WRITE	010	Write stream buffer
SBUF_STREAM_READ_WRITE	011	Read and write stream buffer simultaneously

respectively. The number of SBUF cells in each core, the number of stream buffers between each two adjacent processors as well as the size of the stream buffers can be software configured.

4.2.3 Arbiter

Every type of memory unit has arbiters, as for multi-bank memory units, such as data memory and shared register file, each bank has its own arbiter. In the proposed multi-core architecture, an arbiter arbitrates multiple accesses to a certain memory unit from different cores or the same core. For example, when one processing core issues an access request to a shared memory bank which is dealing with other access requests, this request will be pending and put into a waiting queue in the arbiter. The duration of the pending period is difficult to be predicted during compile time. If an arbiter receives multiple requests simultaneously, it will use predefined priorities or configurable bits of instructions (e.g. MULTIPTBK_REG_FILE) to decide the execution order. Due to the unknown run-time memory access latency, an arbiter signals certain cores to stop or resume RRC counting if access requests from these cores are pending or completed. Therefore RRCs can incorporate the run-time delays in the total execution time. Meanwhile this signal approach does not allow the follow-up instructions to be executed before the access instructions they depend on finish. It can avoid wrong results and deadlocks. In addition, arbiters for shared data memory and register file support atomic operations, the details of which are explained in Section 4.3.

4.2.4 Crossbar switch

The proposed multi-core architecture supports up to 16 shared data memory banks. The width of each bank is 8-bit. As shown in Table 4.1, RMEM and WMEM interface cells can be used to access up to 32-bit data. Therefore, it is necessary to have devices like crossbar switches to route the access request to the proper banks. Usually, a RICA processor has four

RMEM and four WMEM interface cells. In the proposed multi-core architecture, each RMEM or WMEM interface in a processing core connect to all banks through a 4 x *m* crossbar switch, where *m* is the number of banks. According to the value of *m*, the referenced address and the accessing mode (e.g. single word access and half word access), each memory access request to the shared data memory can be routed through switches to the proper banks.

4.2.5 Interrupt controller

Each processing core has one local interrupt controller which controls interrupt requests from different interrupt resources. All interrupt resources connect to interrupt controllers through dedicated interrupt channels. Each time an interrupt controller only sends the interrupt request with the highest interrupt priority to its coupled processing core. In each interrupt controller, there are three memory addressed registers: interrupt status register, mask register and interrupt service register. Each core can access these registers in its local interrupt controller via memory addresses. Currently, the multi-core architecture supports two kinds of interrupts, new task request interrupt and semaphore release interrupt. Both of these are Inter-Processor Interrupts (IPIs) which are sent from one core to another core. The new task request IPI can only be sent by slave cores to the master core. When a slave finishes its current task, it sends this IPI to the master, requesting a new task. Then the master runs a corresponding interrupt handler to dispatch the information of a new task to that slave. The semaphore release interrupt happens when a processing core releases a semaphore used to protect the shared data. The details about this IPI are given in Section 4.3.

4.2.6 Router

The master processing core can set registers in its interrupt controller and task related registers in slave cores through a router. All these registers are memory mapped. This router has an address-to-port map. According to the addresses sent by the master, the router can deliver the information to the proper components through ports connected to them.

4.3 Synchronisation methods and atomic operation support

In order to avoid race conditions caused by simultaneous accesses to a shared resource from different processing cores, two inter-processor synchronisation methods, spinlock and binary semaphore, have been implemented. In the multi-core context, a process should require and release a spinlock or a semaphore before and after accessing the shared data, respectively. A spinlock is a lock where a process repeatedly checks the availability of the lock - a method called busy-waiting. In other words, if a spinlock required by a process is not available, the process will keep asking for this lock until it is released by one of the other processes that previously acquired it. A spinlock allows only one process to access the shared resource protected by the lock at any given time. Spinlocks are efficient when the waiting period is short, as they can avoid overheads introduced by interrupt handling and context switching. However, busy-waiting based spinlocks cause more power consumption and access competition to the shared memory units where locks are stored. If the waiting processing core is blocked for a long period, a binary semaphore is a more efficient approach to implement the synchronisation. In this thesis, similar to spinlocks, binary semaphores make protected shared resources available to only one process at any given time. But the difference is that if a semaphore required by one process is unavailable, the process will go to sleep and the core running this process will do nothing but wait. When the semaphore is released by another process, the releasing process will send a signal to all waiting processes through a semaphore release IPI, and inform the waiting ones that the semaphore is now available. Otherwise, the waiting processes would not know the availability of the semaphore, because they do not have a busy-waiting or regular check scheme. Then the waiting processes will wake up and reissue a request for the semaphore, in case that there are more than one processes waiting for the semaphore. This suspend-wakeup based semaphore synchronisation method is broadly used in many operating systems [107]. The only major difference is that the waiting processes are waked up by the OS instead of the process releasing the semaphore. By using this semaphore approach, memory access competitions caused by busy-waiting based spinlocks, which may take up a reasonable ratio of the overall execution time, are eliminated.

Both spinlocks and semaphores are defined as global variables stored in shared memory blocks. The requisition of a spinlock or a semaphore needs the support of atomic read-modify-write operations. During the atomic operation, the value of a synchronisation variable (either spinlock or semaphore) is guaranteed to be read, modified and written back without any intervention by operations from other processing cores. An atomic read-modify-write operation can be implemented by a single instruction such as test-and-set or compare-and-swap. However, the use of a pair of special instructions – Load-Link and Store-Conditional (LL/SC) - is seeing increasing popularity [108]. In this thesis, LL/SC instructions are used to implement atomic operations and therefore synchronisation methods. The idea behind LL/SC is that LL loads a synchronisation variable into a register, and is immediately followed by an instruction that manipulates the variable. In this thesis, this instruction is subtracting one. Then SC tries to write the variable back to the memory location if and only if the location has not been written by other processing cores after LL was completed. A pseudo-assembly code for a spinlock and a semaphore implemented by LL/SC is shown below:

```
Acquire: LL      reg1, location          // LL the variable's value to reg1
         BEQZ    reg1, Acquire/Sleep     // if unavailable, try again or sleep
         SC      location, reg2          /SC the variable's new value back
         BEQZ    Acquire                 //failed, try LL again

Release: WRITE   location,    1          //write 1 to location
Sleep:   NOP                             // do nothing
```

In the proposed multi-core architecture, for implementing LL/SC, each arbiter for shared memory blocks contains a flag and an address register for each processing core. When an LL instruction from one processing core reads a synchronisation variable, the corresponding

Table 4.6 Access modes and configuration bits for LL/SC instructions

Mode		Configuration bits	Description
LL	LL_SI	1001	Load link a single word
	LL_SE_HI	1010	Load link a half word with sign extension
	LL_ZE_HI	1011	Load link a half word with zero extension
	LL_SE_QI	1100	Load link a quarter word with sign extension
	LL_ZE_QI	1101	Load link a quarter word with zero extension
SC	SC_SI	100	Store conditional a single word
	SC_HI	101	Store conditional a half word
	SC_QI	110	Store conditional a quarter word

flag is set and the variable address is stored into the corresponding address register. Whenever the referenced address of a write request (either normal write operation or SC) is matched against the value of the address register, the flag is reset. An SC instruction enables a check of the flag. If it has been reset, meaning that an intervening write operation occurred before, the SC fails, otherwise, the SC succeeds. Table 4.6 describes access models and configuration bits for LL/SC instructions. The original RICA processor does not support atomic operations. Hence LL and SC instructions are temporarily represented by normal RMEM and WMEM instructions in inline assembly code. It is because LL/SC instructions have the same interface and pins as normal memory access instructions. Later in the trace file, these temporary instructions will be modified to LL/SC instructions with proper configuration bits, as shown in Table 4.6. These atomic instructions can be interpreted and executed by MRPSIM simulator to be described in Chapter 5.

4.4 Custom instruction integration

The existing RICA tool-flow provides full support for the inclusion of both combinatorial and synchronous custom instruction cells through simulator libraries. As shown in the custom cell generation environment of Figure 4.5, the function descriptions of custom cells such as MULTIPTBK_REG_FILE and SBUF are written in C++ via template classes

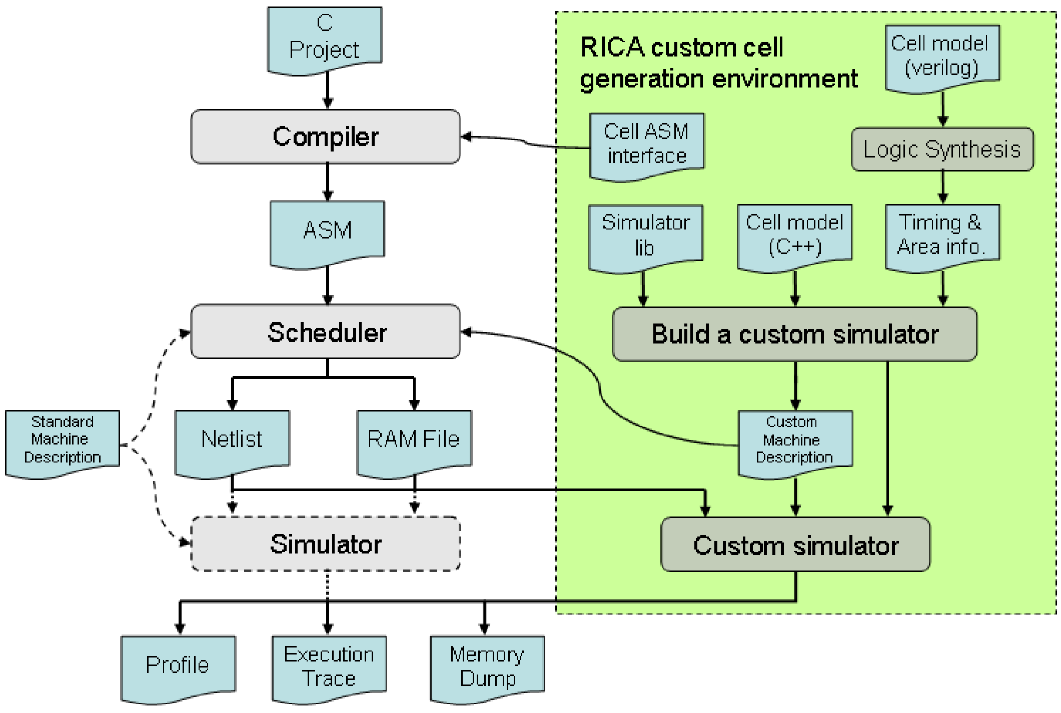


Figure 4.5 RICA custom cell generation environment and custom tool flow

provided by the RICA simulator. A fully automated system generator compiles standard RICA simulator libraries together with the custom cell C++ model and the timing & area information attained by synthesising the custom cell Verilog model. A custom MDF and a custom simulator are generated to replace the standard MDF and simulator used in the standard tool flow. Meanwhile the custom cell assembly interface is provided to enable programmers use custom instructions as inline assembly in their programs.

4.5 Summary

This chapter introduced a master-slave based multi-core architecture using a coarse-grained dynamically reconfigurable processor – RICA. This architecture provides a variety of memory architectural options for different application requirements. These options include shared/local data memory, shared register file and stream buffer. For dealing with the competitions for these shared storage resources, arbiters have been developed to arbitrate the access requests. Crossbar switches were used for routing memory access requests from processing cores to proper shared data memory banks. Meanwhile each processing core has an interrupt controller managing inter-processor interrupts, while the master core owns a router unit which conveys information from the master core to its interrupt controller and slave cores. In addition, two synchronisation methods (i.e. spinlock and semaphore) have been developed for inter-process synchronisation. A pair of atomic options LL/SC was used to implement the synchronisation methods. Furthermore, a custom cell generation environment and a custom tool flow were introduced for integrating custom instruction cells into the RICA architecture.

The next chapter will present a SystemC based simulator which models this proposed multi-core architecture. Based on this architecture, Chapters 6 and 7 will focus on homogeneous and heterogeneous multi-core solutions for WiMAX, respectively.

Chapter 5
Multiple Reconfigurable Processors Simulator

5.1 Introduction

As silicon process technologies shrink to the deep submicrometer region, more transistors are allowed to be integrated into a single chip. As a result, computer architects are able to build more sophisticated designs like multi-core processors. Intel's Quad-Core Xeon processor [35] is a practical example of this, based on 45nm and containing 820 million transistors. Likewise, ever larger and more complicated applications are being developed, fuelled by the abundant resources on multi-core processors. This is also true for embedded systems where large and complex real-time applications like wireless communication and multimedia are targeted. For fast verification of such high-end applications at an early design stage, there is a high demand to produce rapid and accurate simulation tools for modelling the underlying complex multi-core systems. Meanwhile, for developing efficient multi-core solutions for certain applications, a simulation tool is the key for the design space exploration.

In last chapter, a RICA core based multi-core architecture has been presented. For modelling this architecture and verifying applications on it, this chapter introduces a fast, flexible and cycle-accurate simulation tool, called Multiple Reconfigurable Processors Simulator (MRPSIM). MRPSIM is based on SystemC transaction-level modeling [109]. As a higher abstraction level, TLM based simulation is much faster than RTL simulation. TLM enables concurrent hardware/software co-design for early software development, fast architecture modeling and functional verification. In MRPSIM, the functionality of the program is decoupled from the timing simulation. This is done by feeding MRPSIM the execution trace file generated from a cycle-accurate single RICA simulator shown in Figure 3.2.

This chapter is structured as follows. Section 5.2 reviews the existing work on multiprocessor simulation tools. Section 5.3 explains how MRPSIM can provide fast simulation speed and keep accuracy, based on a trace-driven approach. Section 5.4 introduces SystemC and transaction-level modeling, while Section 5.5 describes the details of the TLM model for MRPSIM. Section 5.6 addresses the command line options offered by MRPSIM. Section 5.7 presents a Perl based trace preprocessing tool – Mpsockit which can facilitate the multi-core simulation. Section 5.8 demonstrates the simulation speed of MRPSIM by running a range of test benches on it.

5.2 Related work on multiprocessor simulators

There are a number of multiprocessor simulation tools proposed in both industry and academia, such as those in [110-115]. In [110], a multiprocessor enhancement was proposed for SimpleScalar [116] which is a popular simulation tool in the research community and models a MIPS-like architecture. The authors of [111] presented a simulator called Rsim for shared memory multiprocessors with ILP processors. In [112-114], the approach of decoupling the functionality from timing simulation was used as well, however in a different manner to the approach taken in MRPSIM. Both GEMS [112] and RASE [113] simulators interact with Simics [117], a full system simulator, by feeding timing information to it.

Obviously, the simulation speed can not be fast, since Simics is involved during the multiprocessor simulation. SESC [114] uses instruction streams generated by MINT, an emulator for the MIPS architecture, to model chip-level multiprocessors with out-of-order processors. In [115], a simulation environment called MPARM was proposed based on SystemC RTL and SWARM (software ARM) simulator [118]. However, partially describing the system at a lower SystemC abstraction level, MPARM is inevitably slower than a TLM based simulation tool. Moreover, most of these simulation tools are designed for multiprocessors with conventional processors like ARM and MIPS. To the best of my knowledge, there has been little research done on developing TLM based simulation tools for chip-level multiprocessor using coarse-grained dynamically reconfigurable processors (e.g. RICA).

5.3 Trace-driven simulation

MRPSIM is a trace-driven simulator. The advantage of trace-driven simulation over execution-driven simulation is that once the trace has been collected, it can be reused as a constant input to the simulator when the multi-core architecture varies. However, different from other trace-driven simulators which sacrifice the accuracy for simulation speed, MRPSIM can maintain the timing accuracy with a rapid architecture analysis. The idea is that the functionality of programs and the calculation of static timing are decoupled from the dynamic timing simulation. In my simulation approach, the entire simulation timing is separated into static timing and dynamic timing. Static timing represents the time consumed by the combinatorial critical path in each step and is not affected by the run-time execution. The static timing for each step executed on each core is calculated by the scheduler through using the timing results generated from logic synthesis. This is because each RICA core can be individually synthesised, and that the behaviour of a given core is deterministic due to the RRC. As for dynamic timing, it refers to the time taken by communication instructions (e.g. read and write memory). Basically, these communication instructions are determined during run-time in the multi-core simulation, due to the competition such as from multiple cores, or multiple memory accesses from a same core.

To implement this decoupling approach, firstly the application is partitioned into multiple tasks, and each task is executed by a single RICA processor simulator. The generated execution trace file for each task is fed into MRPSIM simulator. Details of mapping and partitioning are addressed in Section 6.2. Figure 5.1 shows the interface and internal structure of MRPSIM simulator. Execution trace files record the detailed program execution, including all the relevant information necessary for accurate timing, such as static timing and the information for communication instructions. The format for a step in the execution trace file is shown in Table 5.3 in Section 5.7. In Table 5.3, the value of RRC contains static timing in terms of RRC cycles for the current step. The trace parser within MRPSIM converts these machine-specific execution trace files into a machine-independent intermediate representation that captures the static timing model and flattened program control flow structure. Then, the dynamic delays imposed by memory access, synchronisation and IPIs are obtained from the run-time behaviour of the TLM model. Since execution traces already include static timing, MRPSIM only models the behaviour of communication instructions, which contribute to dynamic timing. Computation and control

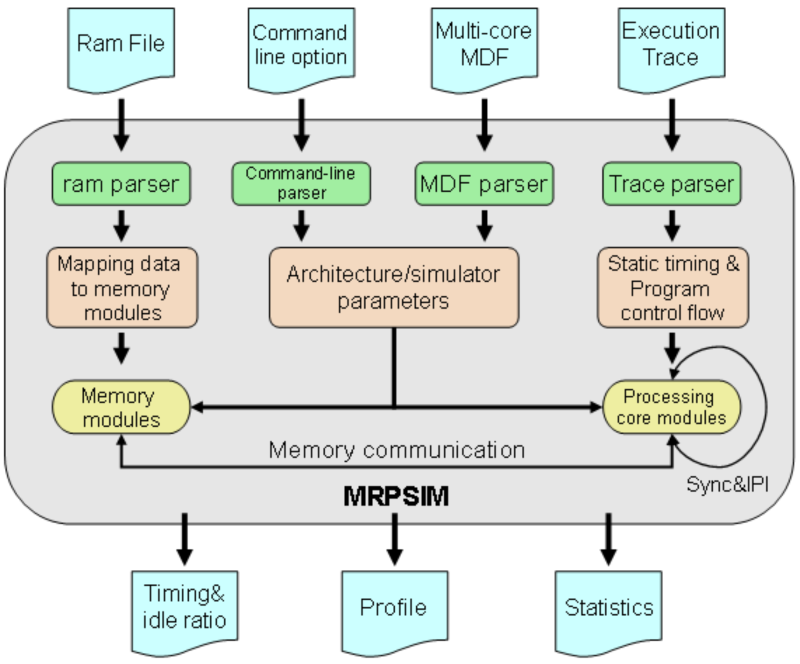


Figure 5.1 The interface and internal structure of MRPSIM simulator

instructions (e.g. addition and jump) are not executed in MRPSIM. In other words, MRPSIM only needs to model the additional overhead incurred during run-time in multiprocessing environments. These overheads include issues such as arbitrating the memory access conflicts. This computation/communication decoupling approach has popularly been used in many other research studies on cycle-accurate simulators such as GEMS [112].

Other inputs of MRPSIM include ram files, multi-core MDF and command line options. Ram files describe the layout of data symbols (e.g. synchronisation variables, shared/local data) in the various data memories, and are mapped onto modelled hierarchical memory modules. The MDF contains architectural information such as memory size and access delay for both local and shared memory as well as RRC period. In addition to architectural parameters in MDF, the executable file of MRPSIM is highly parameterisable by command line options which are described in Section 5.6. All the parameters extracted from MDF and command line options are used to set modules of processing cores and memory modules in MRPSIM.

After performing simulations with MRPSIM, generated performance results are used as feedback to change the design strategy such as the task partitioning method and architecture parameters in order to achieve better performance. Besides generating basic performance information such as timing, memory access count, and processing core idle ratio, MRPSIM can produce advanced information like statistics and processor profiles. In MRPSIM, there is a built-in statistics function for each fundamental module (e.g. processing cores, memory units, and arbiter units). These statistics functions dump out raw statistics, for instance, the detailed memory access information for each individual memory bank, register file bank, and processing cores. These statistics have proven to be vital in analysing simulation results and therefore the design space exploration for multi-core architectures. The processing core profile records the starting time and the finish time for each step, and the request time and finish time for each access to memory, register files and stream buffers. The profile information is crucial for both performance analysis and debugging.

5.4 SystemC and transaction-level modeling

As more and more functions are integrated in SoCs, the current SoC design faces unprecedented challenges in terms of explosive design complexity, time-to-market pressure and increasing cost. However the classic SoC design flow is not the right answer to these challenges [119]. In the traditional SoC design flow shown in Figure 5.2, a system specification is partitioned into hardware development and software development, of which different teams take the responsibility, respectively. There is little cooperation between the two separate development paths, until either an FPGA prototype or a test chip is available. Therefore any hardware or software error found in the system validation stage would result in a new iteration of redesign which dramatically slows the time-to-market and increases the cost. Meanwhile in the traditional hardware development, there exists another design bottleneck. Due to the lack of a common environment, system designers write a conceptual model of the hardware system in a high level language (e.g. C or Matlab) for verifying the functionality, then hand over the model to RTL designers who later re-implement the hardware system in a Hardware Description Language (HDL) according to this model. Obviously, this transfer process is prone to error. Moreover, it usually happens that some

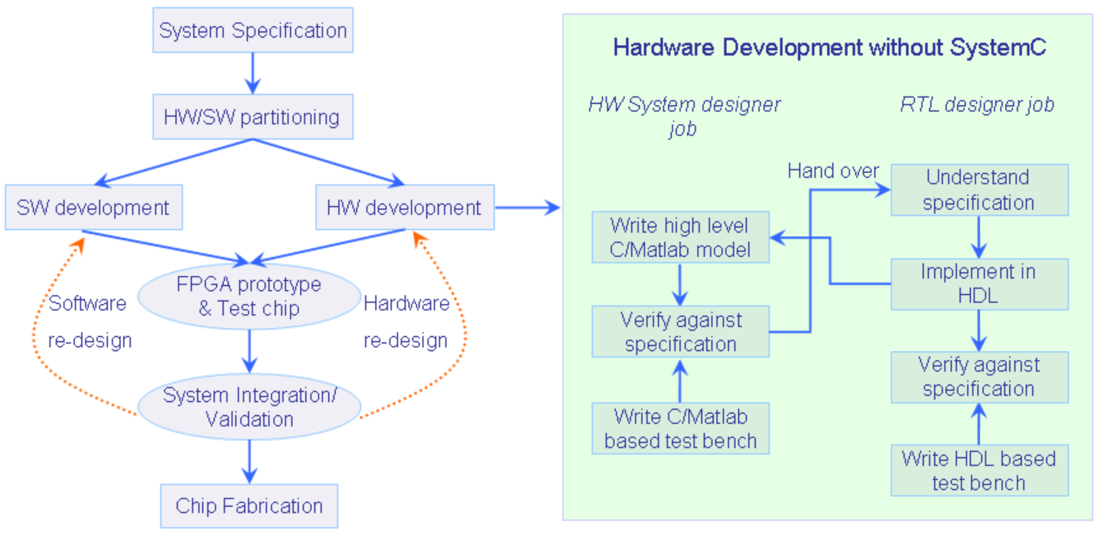
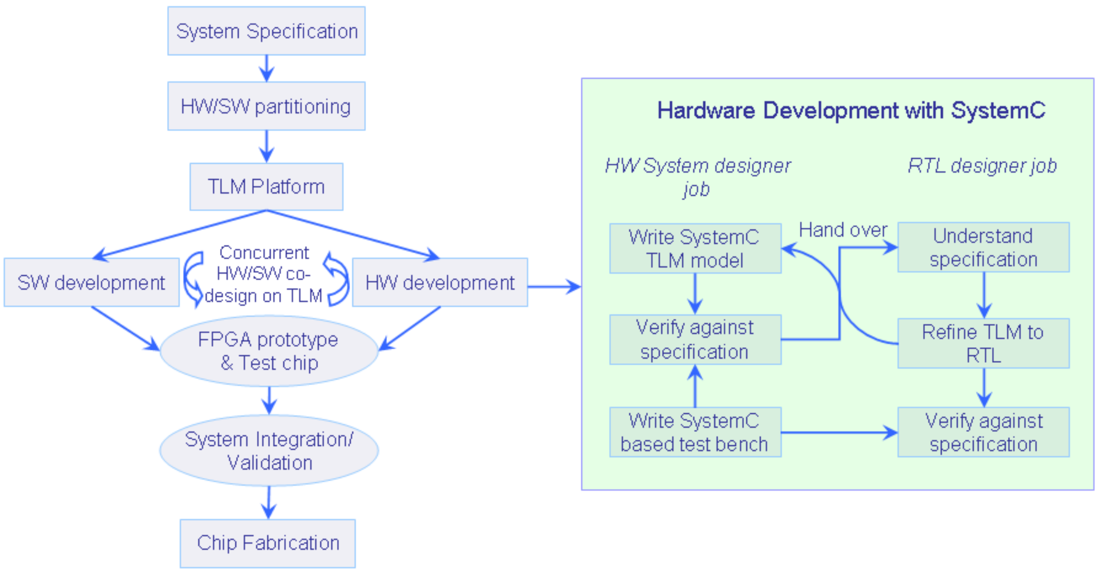

System Specification
HW/SW partitioning
SW development
HW development
FPGA prototype & Test chip
Software re-design
Hardware re-design
System Integration/ Validation
Chip Fabrication
Hardware Development without SystemC
HW System designer job
RTL designer job
Hand over
Understand specification
Write high level C/Matlab model
Implement in HDL
Verify against specification
Write C/Matlab based test bench
Verify against specification
Write HDL based test bench

Figure 5.2 Traditional SoC design flow without SystemC

part of the conceptual model cannot be implemented in RTL, and modifying this model is needed. Hence a costly redesign iteration is involved, since the different teams use two different languages. Due to these inherent design bottlenecks of the traditional SoC design flow, in recent years, a new SoC design flow has become popular [119, 120]. As shown in Figure 5.3, this design flow enables hardware/software co-design by raising abstraction level from RTL to TLM and provides a SystemC based common design environment for both hardware system designers and RTL designers.

SystemC [120] is a System Description Language (SDL) as well as a hardware description extension of C++. By adding a library of special classes, SystemC brings hardware design concepts, such as hardware timing and concurrent processes, into C++. SystemC is fully compatible with the standard C++ programming environment, and thus a SystemC model can be compiled as a normal C++ program. As an open source programming language, SystemC is free to be used and modified. It is a great advantage over its expensive proprietary SDL counterparts such as SpecC. Moreover SystemC supports TLM. TLM [109] is a transaction-based modeling approach which separates communication from computation within a system following the concept of "divide and conquer".

In the SystemC based design flow shown Figure 5.3, a TLM platform is built right after



System Specification
HW/SW partitioning
TLM Platform
SW development
Concurrent HW/SW co-design on TLM
HW development
FPGA prototype & Test chip
System Integration/ Validation
Chip Fabrication
Hardware Development with SystemC
HW System designer job
RTL designer job
Write SystemC TLM model
Hand over
Understand specification
Verify against specification
Refine TLM to RTL
Write SystemC based test bench
Verify against specification

Figure 5.3 SoC design flow with SystemC

HW/SW partitioning, serving as a unique reference for early software development, architecture analysis and functional verification. The TLM platform is presented as an executable file generated by the C++ compiler. TLM enables HW/SW co-design and co-simulation by which hardware and software teams can have a more efficient cooperation in the early phase of the system development. Hence the success probability of the first test chip is significantly increased and time-to-market is shortened [119]. Meanwhile in this design flow, the hardware development is fuelled by using SystemC in both system design and RTL design teams. Based on the same design environment, it is an easier job for RTL designers to refine System TLM model down to SystemC RTL compared to the traditional approach. Furthermore, the SystemC based test bench developed by system designers can be reused in RTL verification.

5.5 TLM model

Due to the advantages of SystemC TLM described in the last section, MRPSIM simulator is implemented in SystemC TLM. Usually, in TLM, system components are modelled as SystemC modules which contain a number of concurrent SystemC processes and/or functions representing their behaviours. The communication among modules is achieved by using transactions through abstract channels which implement TLM interfaces. As the kernel of TLM, TLM interfaces can be accessed by SystemC processes through module ports [119]. In MRPSIM simulator, those components introduced in Chapter 4.2 are modelled as three kinds of SystemC modules, initiators, targets and their combinations. An initiator contains SystemC processes initiating transactions to the target modules which respond to these transactions. Components modelled as initiators include arbiters and interrupt controllers, while memory units, crossbar switches and the router behave as targets. As for processing cores, they are initiators as well as targets. For example a processing core can issue a memory access transaction to memory units through arbiters and respond to interrupt transactions from its interrupt controller as well. Figure 5.4 shows a TLM example where two processing cores can initiate write and read transactions to a memory bank through an

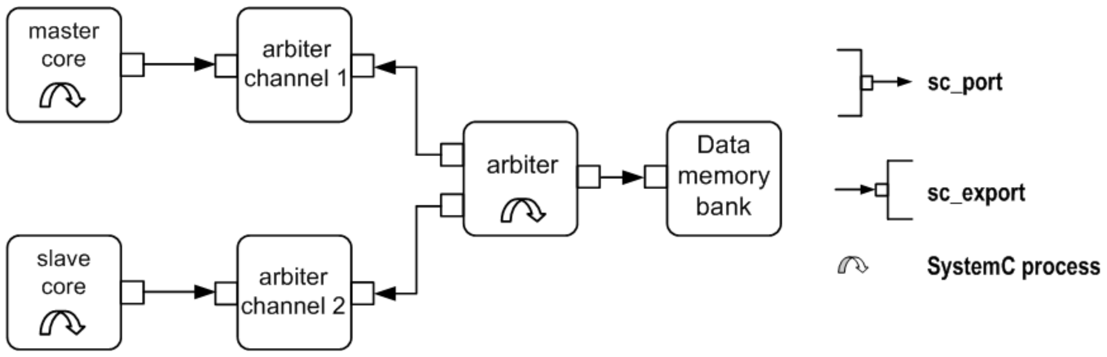


Figure 5.4 An example demonstrating the TLM model of a multi-core architecture

arbiter. As shown in Figure 5.4, modules connect each other by binding sc_port of an initiator to an interface provided by sc_export of a target. In SystemC, binding of all ports to interfaces is done during the elaboration phase and cannot be changed during simulation. In write and read transactions, processes in processing cores put requests into their respective arbiter channels and wait for corresponding responses. Processes in the arbiter scan all access request ports, decide which request has the highest priority and forward it to the memory bank. When the memory bank makes a response, the arbiter puts the response to the relevant channel. Finally, the corresponding core collects the response and completes the transaction.

Table 5.1 shows the functionality of main processes and functions in MRPSIM SystemC modules. All these modules are defined as C++ classes and instantiated in the MRPSIM program main function which incorporates various parsers as well. Meanwhile, modules of processing cores instantiate submodules for communication instruction cells (i.e. WMEM, RMEM, MULTIPTBK_REG_FILE and SBUF). The cell instance count can be read from the MDF. In target modules, functions like read() and write() are called by the transaction initiators through sc_export to implement the corresponding transactions. As shown in Table 5.1, those processes and functions followed by module names in parentheses only appear in those particular modules.

Table 5.1 Functionality of main SystemC processes and functions in MRPSIM modules

Master/Slave core		
Process	boot (master)	Initialise registers in slave cores; activate the run process in slave cores
	run	The main process executing every steps
	pfu	Pre-fetch the next step if prefetching conditions satisfied
	rrc	Reconfigurable rate controller
	pauseRRC	Pause or resume RRC according to signals from arbiters
	bottomHalfJob (master)	Calculate and send a new task information to slave cores which sent new task interrupts
Function	isrJob (master)	Rapid response to new task interrupts from slave cores; activate the bottomHalfJob process
	read (slave)	Read registers, called by the master core through sc_export
	write (slave)	Write registers, called by the master core through sc_export
Data/program memory/Register file/Stream buffer		
Function	initialise (data/program memory)	Initialise data/program memory by using data/instruction traces extracted from ram/trace files
	read	Read memory blocks, called by processing cores through sc_export
	write	Write memory blocks, called by processing cores through sc_export
Arbiters for Data memory/Register file/Stream buffer		
Process	scan	Scan all connected access request ports and find access requests
	run	After scanning, forward the highest priority request to the memory bank or activate read and/or write processes
	pauseRRC	Send signals to relevant cores to pause or resume RRCs, if access requests from these cores are pending or completed
	read (reg. file)	Forward read requests to the register file bank
	write (reg. file)	Forward write requests to the register file bank
Function	addInterface	Insert an access request port into a multimap variable
Switch		
Function	transport	Forward the access request to proper shared data memory bank
Interrupt controller		
Process	run	Forward the highest priority interrupt to the processing core
Function	addInterface	Insert an interrupt request port into a multimap variable
Router		
Function	transport	Forward the information to master’s interrupt controller or slave cores

5.6 MRPSIM command line options

MRPSIM has a variety of command line options relative to architecture, performance analysis, debugging and so on. By setting architecture options, the multi-core architecture is highly parameterisable in terms of the number of slave cores, the number of data memory banks, synchronisation methods, single core or multi-core model and so on. For example, in single core mode, MRPSIM can model a single RICA processor and produces the same timing accuracy as the cycle-accurate execution-driven simulator for RICA processor. Performance analysis options control the output of advanced performance information. While debugging options generate debugging information to assist verifying the program functionality and debugging errors. These options are listed in Table 5.2.

Table 5.2 Command line options of MRPSIM

Architectural Option	Description
--arch *type*	Set the type of the multi-core architecture, accepted values of *num*: 0: SPMD mode; 1: master-slave shared memory mode
--bank *num*	Specify the number of banks for the shared data memory
--dmem *type*	Set the type of data memory architecture, accepted values of *type*: 0: share only; 1: share and local
--slave *num*	Specify the number of slave cores, when *num* set to 0, MRPSIM is in the single core mode
--sync *type*	Set the type of synchronisation, accepted values of *num*: 0: lock; 1: semaphore
--pmem *type*	Set the type of program memory architecture, accepted values of *type*: 0: share only; 1: local only
--sbuf-args *nums*	Specify the number of stream buffer banks and the size of each bank
--source-args *files nums*	Specify *files* as source files and the input bit width for each source file
--sink-args *files nums*	Specify *files* as sink files and the output bit width for each sink file
Performance analysis Option	**Description**
--enable-prof *bool*	Enable/disable generation of processing core profile files
--enable-stat *bool*	Enable/disable generation of statistics files
--enable-var-access *bool*	Enable/disable writing the access number for each global variable to var_access.dat
--prof *file*	If --enable-prof is true, write profile to specified *file*
--stat *file*	If --enable-stat is true, write statistics to specified *file*
Debugging Option	**Description**
--cout_mode *mode*	Set the mode of execution information output, accepted values of *mode*: 0: brief; 1: verbose
--dump *file*	If --enable-dump is true, write memory dump to *file*
--enable-dump *bool*	Enable/disable generation of memory dump files
--parser-debug *bool*	Enable/disable debugging output from the trace file parser
--scanner-debug *bool*	Enable/disable debugging output from the trace file scanner
--start-display *start-time*	Specify the time starting to display execution information, unit is ns
Other Options	**Description**
--end-time *end-time*	Specify the time the simulation stops, unit is ns
--mdf *file*	Specify *file* as the input MDF
--execute *num*	Specify how many times programs shall be executed
--per-step *mode*	Set the mode of the trace file parser, accepted values of *mode*: 0: parse the entire trace file once; 1: parse step by step
--trace-format *format*	Set the format of the trace file, accepted values of *format*: 0: each trace file stores the instruction trace of one task; 1: all task instruction traces stored in one trace file

5.7 Mpsockit - a subsidiary tool

Written in Perl, a subsidiary tool called Mpsockit is developed to preprocess the input files of MRPSIM simulator and facilitate the multi-core simulation. One of the most important features of this tool is compressing execution trace files. Usually, an execution trace is huge – sized hundreds of MB for a complex program. This tool can compress trace files by removing instructions which will not be executed by MRPSIM. After compression, the size of traces can be cut down to less than 1/3, on average. As a result of this, the time for parsing trace files by MRPSIM is dramatically reduced. Hence the simulation speed is improved, since parsing the traces constituted a significant part of the entire simulation time. The formats of both original and compressed traces are shown in Table 5.3 for one single

Table 5.3 Formats for original and compressed traces

Original Step Format
Step N: // step index
MULTIPTBK_REG_FILE[0] {.conf = `MPREGFILE_RDNLY; .read_address = 2; .write_address = OPEN; .write_data = OPEN; .out = 10;}
MULT[0] {.conf = `MUL_SIG_SI; .in1 = 5; .in2 = REG[0].out = 10; .out = 50;}
RMEM[0] {.conf = `RMEM_SI, .in_addr = 100; .out = 5;}
RMEM[1] {.conf = `RMEM_SI, .in_addr = 104; .out = 10;}
ADD[0] {.conf = `ADD_ADD_SI; .in1 = RMEM[0].out = 5; .in2 = MULT[0].out = 50; .out = 55;}
ADD[1] {.conf = `ADD_ADD_SI; .in1 = RMEM[1].out = 10; .in2 = MULT[0].out = 50; .out = 60;}
WMEM[0] {.conf = `WMEM_SI; .in_addr = 108; .in = ADD[0].out = 55;}
COMP[0] {.conf = `C_COMP_EQ; .in1 = ADD[0].out= 55; .in2 = ADD[1].out = 60; .out = 0;}
JUMP[0] {.conf = `JUMP_IF_EQZ; .addr_in = 2; .cond = COMP[0].out = 0;}
RRC {.conf = 3;} // cycle count for static timing
Compressed Step Format
Step N: // step index
MULTIPTBK_REG_FILE[0] {.conf = `MPREGFILE_RDNLY; .read_address = 2; .write_address = OPEN; .write_data = OPEN; .out = 10;}
RMEM[0] {.conf = `RMEM_SI, .in_addr = 100; .out = 5;}
RMEM[1] {.conf = `RMEM_SI, .in_addr = 104; .out = 10;}
WMEM[0] {.conf = `WMEM_SI; .in_addr = 108; .in = ADD[0].out = 55;}
RRC {.conf = 3;} // cycle count for static timing
Cell {.in = 9;} // the number of instruction cells used in current step

step. After compression, a new entry called *Cell* is generated, which records the number of instruction cells in the original step, and is utilised for calculating the configuration latency for this step. This tool can also be used to set parameters in MDF, such as how many bytes global variables take, through extracting the information from source files. Other important features of this tool include preprocessing trace files according to different task mapping methods, such as merging multiple traces into one trace for task replication method. Furthermore, as mentioned in Chapter 4, this tool can modify the configuration bits of those temporary instructions for LL/SC which can be interpreted and executed by MRPSIM. The options of Mpsockit and their individual description are given in Table 5.4.

Table 5.4 Mpsockit options

Options	Description
--ci/--cs	Calculate the number of instructions/steps in each input trace files
--merge *files*	Merge multiple trace *files* into one trace for task replication method
--move *files*	Move some instructions from one trace to others for loop partitioning method
--shmem *source mdf*	Get information from *source* to indicate the size of global variables in *mdf*
--split *file*	Split one trace *file* into multiple traces for loop partitioning method
--stack *source mdf*	Get information from *source* to set the stack top for each core in *mdf* for the mapping without local memory
-t *files*	Compress trace *files* Calculate the number of used instruction cells for each step Modify the configuration bits for atomic operations

5.8 Results

For evaluating the performance of MRPSIM, several applications were used as test benches, including an Advanced Encryption Standard (AES) application, a 64-tap Finite Impulse Response (FIR) filter, a 64-point 6-stage radix-2 FFT, an image smoothing and edge enhancement application and WiMAX BPSK based transmitter and receiver applications. An Intel Core 2 2.4GHz PC was used as the host machine for MRPSIM simulator. All

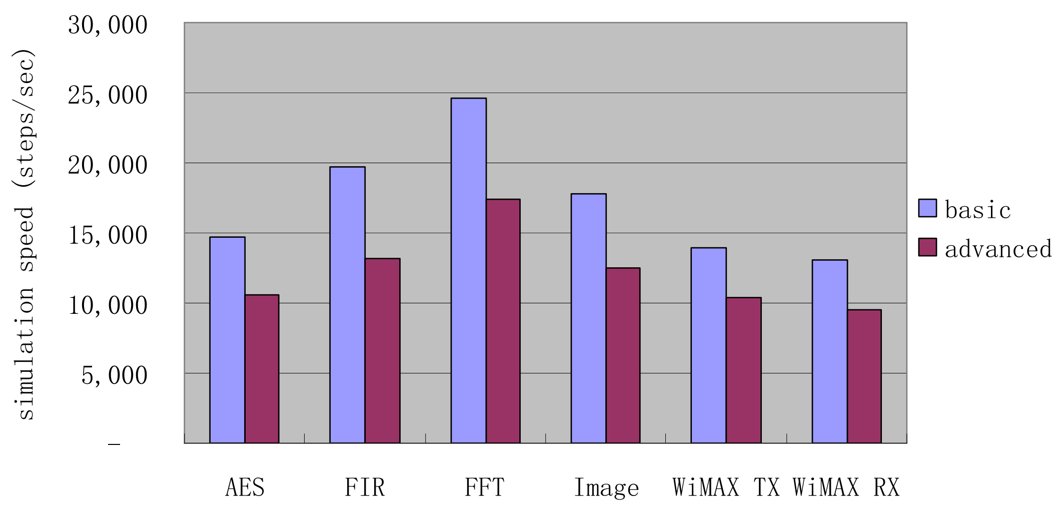


Figure 5.5 The simulation speed (steps/sec)

applications were developed for dual-core architectures with 2ns RRC period. Figures 5.5-5.7 show the simulation performance in terms of three different metrics including Steps Per Second (SPS), Instructions Per Second (IPS) and Cycles Per Second (CPS). The metric of CPS is used to estimate how fast the cycle-accurate model can achieve, from the hardware system point of view. While IPS is a measure of how fast the application can be executed on the simulator, and is a software developers' concern. As for SPS, it is particularly defined for RICA and represents how fast the simulator can execute configuration contexts for this DR processor.

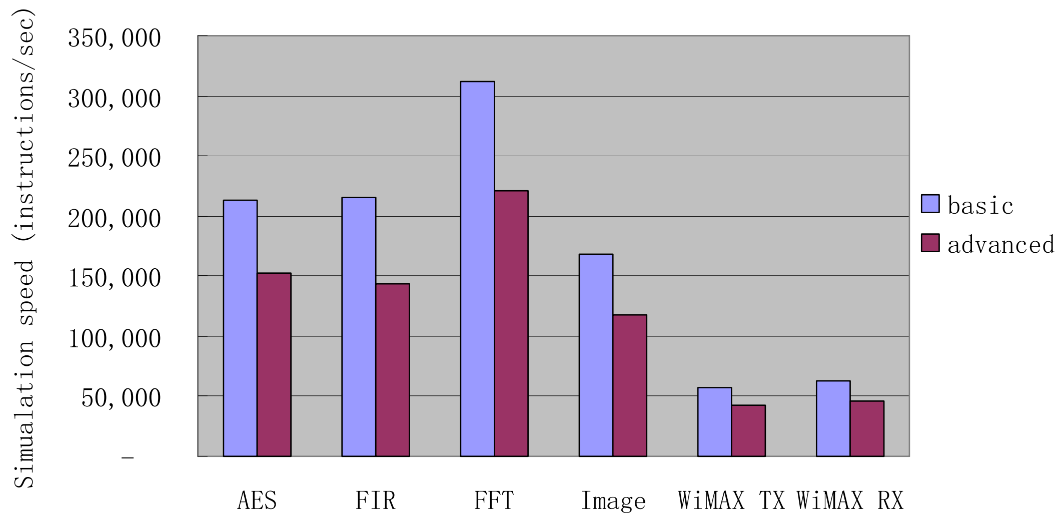


Figure 5.6 The simulation speed (instructions/sec)

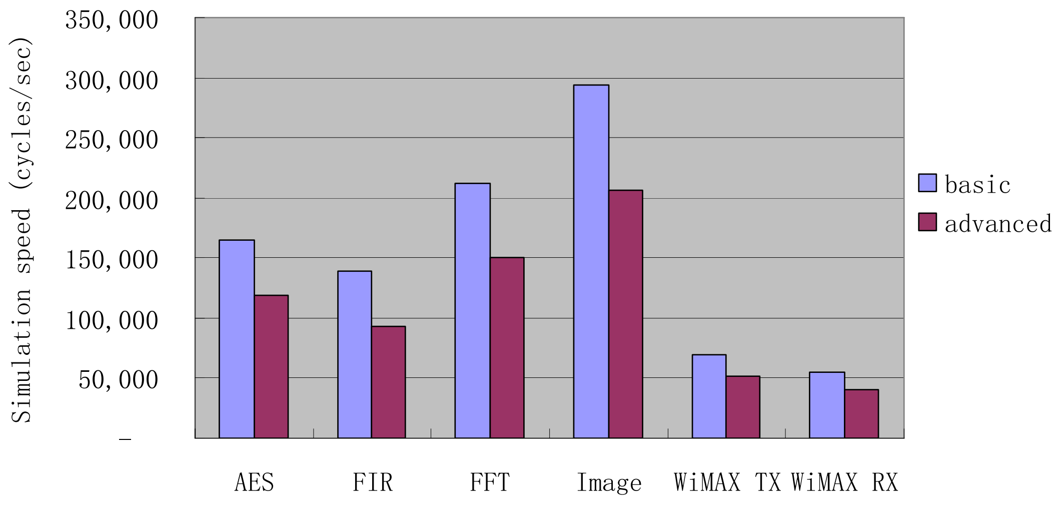


Figure 5.7 The simulation speed (cycles/sec)

Each application owns two sub-test cases, one of which enables MRPSIM to generate detailed statistics and processor profiles, while the other only produces the basic performance information. As shown in Figures 5.5-5.7, MRPSIM simulator can achieve simulation speeds up to 25 KSPS, 300 KIPS and 300 KCPS, respectively, depending on the application. It proves that TLM based MRPSIM simulator can provide much higher simulation speeds, compared to RTL approaches which usually run at several hundred CPS. For the image application, the simulator demonstrates the highest CPS and lowest IPS. It is because the image application involves much more memory access operations which take more waiting cycles due to competition. The simulation of WiMAX applications is significantly lower in terms of IPS and CPS, compared to other applications. It is because WiMAX applications contain many conditional statements which cause frequent context switches and relatively less instructions within each step. Figures 5.5-5.7 also indicates that the collection of detailed statistics and profiles causes about 30% drop in the simulation speed. Figure 5.8 shows the simulation time for each application. Due to the size of the image being processed, the image application takes much longer to run compared to other applications.

5.9 Summary

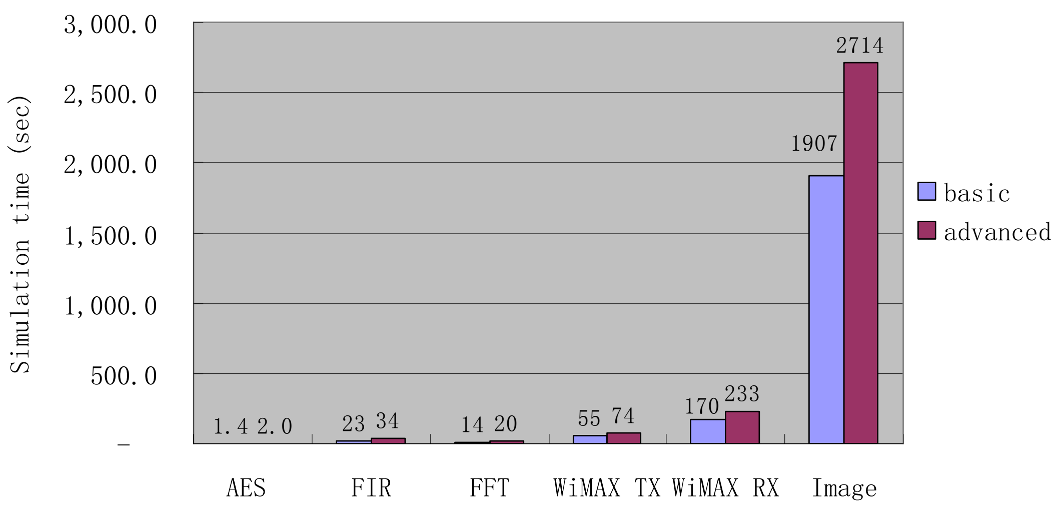


Figure 5.8 The simulation time (sec)

This chapter presented a trace-driven simulator, called MRPSIM, for modeling the proposed multi-core architecture. Implemented by SystemC transaction-level modeling, this simulator provides a variety of architectural parameters and performance information, delivers fast simulation speeds and maintains timing accuracy. MRPSIM can correctly and efficiently simulate the run-time multi-core environment, allowing the throughput of the modelled system to be measured. Meanwhile a trace preprocessing tool, Mpsockit, was introduced to facilitate the multi-core system simulation and task mapping. In addition, several test benches have been executed on this simulator to estimate this simulator's performance. For chosen applications, MRPSIM simulator demonstrated high simulation speeds, up to 300 KCPS, 300 KIPS and 25 KSPS, respectively.

Chapter 6 Homogeneous Multi-core Solutions for WiMAX

6.1 Introduction

Many successful multi-core processors in the market are homogeneous collections of processing cores, such as UltraSPARC from Sun Microsystems and Intel Quad-Core Xeon. One of the salient advantages of homogenous multi-core solutions is that once a core passes the verification test, it can be easily replicated in the multi-core system. Therefore the development time is dramatically shortened. In addition, a homogeneous multi-core system has a simple software model which requires less programming efforts. This chapter introduces several homogeneous multi-core solutions for a fixed WiMAX physical layer. These solutions have different configurations in terms of the number of processing cores, partitioning method and memory architecture. These configurations are chosen for different resource/performance tradeoffs. These solutions originate from the basic multi-core architecture proposed in Chapter 4. All processing cores used in these solutions are based on the RICA architecture. The target WiMAX physical layer uses BPSK modulation and 1/2 rate convolutional coding where puncturing and RS coding are not required. Before describing these solutions, this chapter investigates how multiple tasks can be mapped onto the multi-core architecture and synchronise with each other.

This chapter is structured as follows. Section 6.2 presents a mapping methodology used to partition, schedule and map tasks onto the multi-core architecture. Section 6.3 addresses three task partitioning methods: task merging, task replication and loop partitioning. Section 6.4 introduces how a multi-core project can be developed for the multi-core architecture. Section 6.5 describes how tasks synchronise with each other by using the synchronisation methods introduced in Section 4.3. Section 6.6 describes these homogeneous multi-core solutions and demonstrates results.

6.2 Mapping methodology

Basically, the mapping and scheduling of tasks involved in the implementation of WiMAX on a multi-core architecture are complex optimisation problems. These problems need to be solved simultaneously to maximise the throughput. In addition, data transmission time between different processing cores must also be taken into account during task mapping and scheduling. As shown in Figure 6.1, a mapping methodology is developed to enable running multiple tasks on MRPSIM simulator. This methodology incorporates profiling-driven task partitioning, task transformation and memory architecture aware data mapping in order to reduce the overall application execution time. This methodology allows the designer to explore the different multi-core implementations on MRPSIM simulation platform. During the process of task partitioning and task mapping, execution-time estimation for the different tasks, as well as the time estimation for data transfer between processing cores, are required. It is also necessary to schedule the execution order of pipelined tasks, such as those shown in Figure 2.1, to improve the system performance. The execution time of a task performed by a single RICA processor can be obtained from profiling information generated by the single RICA simulator. The mapping flow starts from the description of an application in standard sequential C code which is optimised for a single RICA implementation. Then a

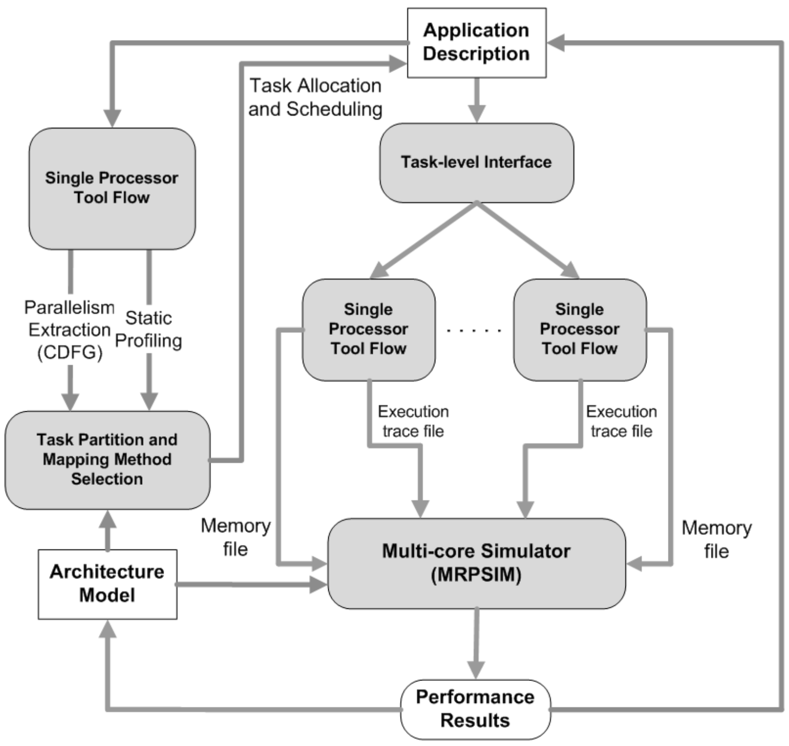

Application Description
Task Allocation and Scheduling
Single Processor Tool Flow
Task-level Interface
Parallelism Extraction (CDFG)
Static Profiling
Single Processor Tool Flow
. . . . .
Single Processor Tool Flow
Execution trace file
Execution trace file
Task Partition and Mapping Method Selection
Memory file
Memory file
Architecture Model
Multi-core Simulator (MRPSIM)
Performance Results

Figure 6.1 Mapping methodology

proper task partition and mapping is chosen after executing the application using the single RICA tool flow.

As shown in Figure 6.1, the task partition and mapping selection is based on the control data flow graph and static profiling generated by the single RICA tool flow. The control data flow graph provides instruction level parallelism information for loop level partitioning. The static profiling information contains the timing characteristics for each task and the access frequency for the various data items. This information is used for mapping tasks to available processing cores as well as mapping various data items to proper memory locations. After the task partition and mapping method is decided, the application is modified to run on a multi-core system. A task-level interface, including the macros for synchronisation, is developed to facilitate the multiprocessing programming. In addition, this mapping methodology supports both homogeneous and heterogeneous architectures. Therefore the resource mix for each processing core in the system is allowed to differ, and may be tailored to the particular tasks that it is intended to execute. Through this method, the multi-core processor model can support both homogeneous and heterogeneous multi-core architectures based on dynamically reconfigurable processor, or even conventional DSPs (or any mixture thereof).

The data mapping stage performs mapping of data items to the memory architecture and explores different memory architectures for minimising memory access latencies. As described in Section 4.2.2, in the proposed multi-core architecture, each processing core can access four types of memory components, shared multi-bank register file, stream buffer, shared data memory and local data memory. A local data memory is private to a processing core and cannot be accessed by other cores. Thus, shared data items, accessed by tasks running on different cores, should not be mapped to these memory blocks. Instead, they can be mapped to shared register file, or shared memory, which can be accessed by the different cores. As shown in Figure 6.1, through each single processor tool flow, trace files and memory files are generated and fed into MRPSIM simulator. Eventually, after one exploration iteration, the architecture model and the application can be modified to pursue higher throughput according to the performance report.

6.3 Task partitioning methods

As shown in Figure 2.1, both WiMAX transmitter and receiver consist of five main functional blocks. For BPSK based WiMAX, convolutional coding is used as FEC encoding. Correspondingly Viterbi encoding is used as FEC decoding. For one OFDM symbol, randomising, convolutional coding, interleaving and modulation jointly have an execution time of 32 μs , while the OFDM downlink processing function takes 200 μs . In the receiver side, there are six functions. Among them, the decoding function (including both Viterbi decoding and de-randomising) takes 1567 μs where Viterbi decoding is the most time consuming function, while other functions (i.e. synchronisation, OFDM uplink processing, demodulation and de-interleaving) totally run 180 μs . The design challenge is to map the application onto an architecture optimised in terms of system performance and cost. This includes not only the mapping strategy but also architectural design choices such as the number of processing cores as well as the number and types of instruction cells in each core. At present, the optimisation involves a lot of manual work. A tool which can make automatic optimisation would be a research target in future.

6.3.1 Task merging and task replication

A simple mapping would assign each function as a task to one processing core for both transmitter and receiver. This mapping generates a multi-core architecture with totally eleven processing cores (i.e. five cores for transmitter and six cores for receiver). Each core acts as one stage of either transmitter or receiver processing chain, as shown in Figure 2.1. However, this partitioning method would lead to highly unbalanced workloads among cores. The overall throughput of the WiMAX application is limited by the most time-consuming tasks, which are the OFDM downlink processing task in the transmitter side, and the Viterbi decoding task in the receiver side. The processing cores with a light workload would spend a considerable time idle waiting for synchronisation. To improve the load balance, merging and replicating tasks can be employed in the mapping strategy. The task merging method incorporates several tasks into one single task which keeps the original execution order of these tasks. This method can reduce the number of processing cores, but requires larger local program and data memory for each core running the merged task. In the transmitter side, the randomising, convolutional coding, interleaving and modulation tasks are merged into one single task called channel coding that sequentially performs these original tasks. The merged channel coding task and the OFDM downlink processing task are assigned to different processing cores, as shown in Figure 6.2 (a). Similarly, for the receiver, the synchronisation, OFDM uplink processing, de-interleaving and demodulation tasks are merged and assign to one core. The Viterbi decoding and de-randomising tasks are merged into one task called Task decoding and assigned to another core, as shown in Figure 6.2 (b). This mapping strategy involves four processing cores, as both transmitter and receiver occupying two cores. Obviously, workloads are unbalanced and throughputs are still determined by the most time consuming tasks (OFDM downlink processing in the transmitter and decoding in

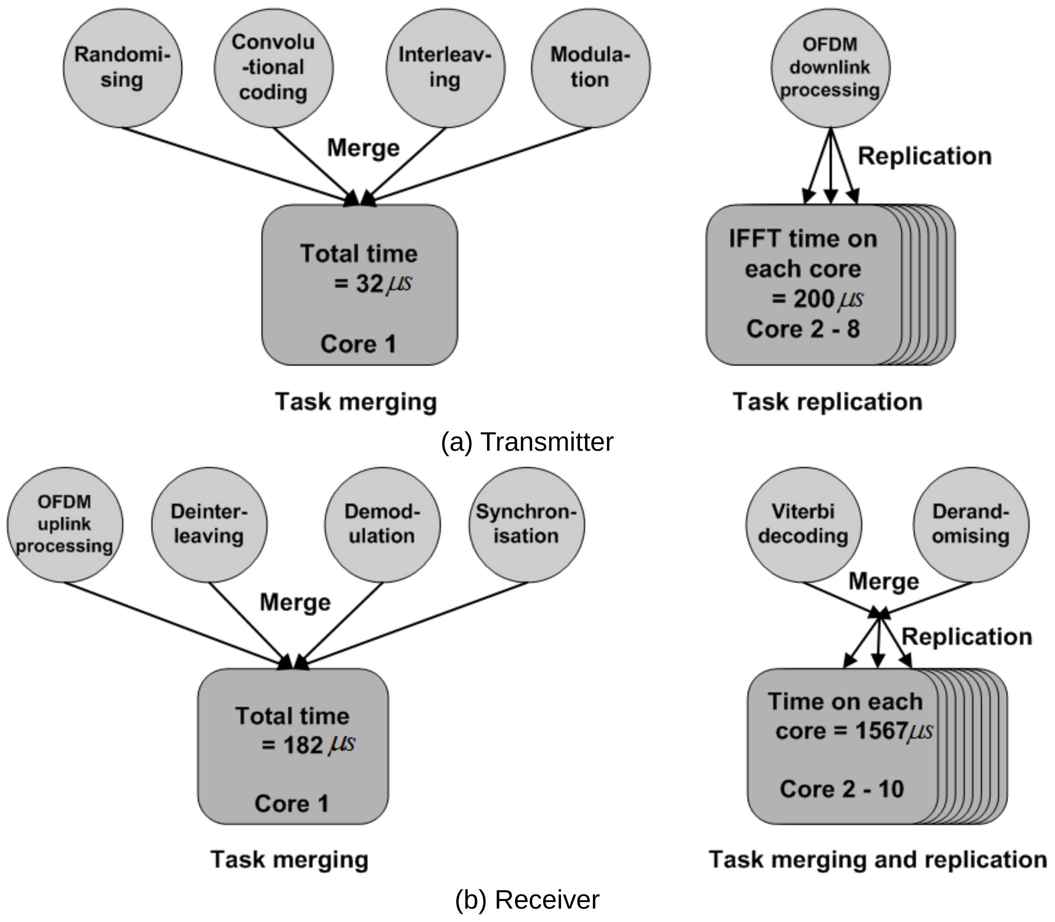


Figure 6.2 Task merging and task replication methods

the receiver).

To further improve throughputs, task replication can be applied to the tasks on critical paths. Task replication assigns the same task to several processing cores such that all instances of the task perform different OFDM symbols in parallel. As a result, task replication needs more processing cores to carry out the parallelism as well as larger shared data memory for storing the shared data within multiple OFDM symbols. As shown in Figure 6.2 (a), in the transmitter, the OFDM downlink processing task is replicated to seven processing cores, as its execution time is about seven times that of the merged channel coding task. Ideally, the OFDM downlink processing task will complete seven parallel instances every 200μs, resulting in a 700% performance improvement. As for the receiver, similarly the decoding task is replicated to nine processing cores. Clearly, the combination of task merging and task replication can lead to higher performance solutions. However, task merging requires more local memory and task replication requires more shared data memory as well as more processing cores. Obviously, for a multi-core system which has limited computing and storage resources, task replication can not be applied.

6.3.2 Loop-level partitioning

As discussed before, for both transmitter and receiver, the task merging with two cores results in an unbalanced workload, leaving the first core idle for a significant fraction of the time. In order to balance the workload among the two processing cores, another partitioning method has been proposed, called loop partitioning which includes both loop splitting and instruction level partitioning. This method divides the tasks at the instruction level instead of the function level in order to explore the instruction level parallelism within a task. In the transmitter side, the overall execution time of all the other tasks is much less than the execution time of the OFDM downlink processing task where the main function is 256-point IFFT. In this thesis, the 256-point IFFT is based on radix-2 FFT algorithm which has eight pipeline stages, each stage involving 128 butterfly operations. The most time-consuming part of the FFT algorithm is the 2-level loop body shown in Table 6.1 where the left hand code is the original code running on a single processor. Originally as a compiler optimisation technique, in this thesis, loop splitting is used to break a loop into multiple loops which then are assigned to different processing cores. The generated loops keep the same original loop body but iterate over different portions of the original index range. Shown as the right hand code of Table 6.1, the outer loop is split into two parts. The first part executes the first four stages, while the second part executes the remaining four stages. The two parts are assigned to two different processing cores.

Table 6.1 IFFT Loop Partitioning

Original Code	Split Code

// the number of stages
#define STAGE 8;
// the size of FFT
#define SIZE 256;
 	
int half_size = SIZE/2;
 	
for (i =0; i < STAGE; i++) {
    counter = 0;
    do {
        butterfly(real, image);
        counter++;
    } while (counter < half_size)
}
	#define STAGE 8; // the number of stages
	#define SIZE 256; // the size of FFT
	/* assigned to Processing core 0*/
	/* load data from shared memory to Core0 local memory*/
	Load(real, image, core0_real, core0_imag);
	for (i =0; i < STAGE/2; i++) {
	    counter = 0;
	    do {
	        butterfly(core0_real, core0_image);
	        counter++;
	    } while (counter < half_size)
	}
	/*store back results from Core0 local memory to shared memory*/
	Load(core0_real, core0_imag, real, image);
	
	/* assigned to Processing core 1*/
	/* load data from shared memory to Core1 local memory*/
	Load(real, image, core1_real, core1_imag);
	for (i =0; i < STAGE/2; i++) {
	    counter = 0;
	    do {
	        butterfly(core1_real, core1_image);
	        counter++;
	    } while (counter < half_size)
	}
	/*store back results from Core1 local memory to shared memory*/
	Load(core1_real, core1_imag, real, image);

In the receiver side, the decoding task is the most time-consuming. Here, the loop partitioning method uses both loop splitting and instruction level partitioning in order to parallelise the resulting code across different processing cores. The decoding task contains the most time-consuming loop body with a loop-carried dependence, which means each iteration of a loop depends on values computed in an earlier iteration. This prevents further efficient partitioning at the task level. As shown in Figure 6.3 (a), the body of the main loop in the decoding task iterates 870 times and includes 16 butterfly operations which are the main operations in Viterbi decoding. A butterfly operation consists of independent and dependent data paths. The independent data paths can be run on different processing cores to increase the instruction level parallelism. To make use of the idle time of the first core, loop splitting can be used to effectively move some work of the decoding task from the second core to the first core running the merged task. To achieve this, the decoding loop body is written in two ways: one with all the work being done on a single processing core, and the other where some of the butterflies are moved to the other processing core, such that they can be performed in parallel, but need to be synchronised to allow for the dependencies to be met. The second core then runs the first (single core) version when the first core is busy executing the merged tasks, then switches to the second version (dual-core), allowing the now idle first processing core to share in the decoding task.

Based on the execution time of the merged task, the loop is split into two parts: the first 840 iterations and the last 30 iterations as shown in Figure 6.3 (b). The execution time of the last 30 iterations is nearly equal to the merged task. The body of the loop in the first 840 iterations is partitioned across two processing cores, as shown in Figure 6.3 (b). After loop partitioning, the merged task and part of the first 840 iterations are assigned to one processing core, the remaining part of the first 840 iterations and the whole of the last 30

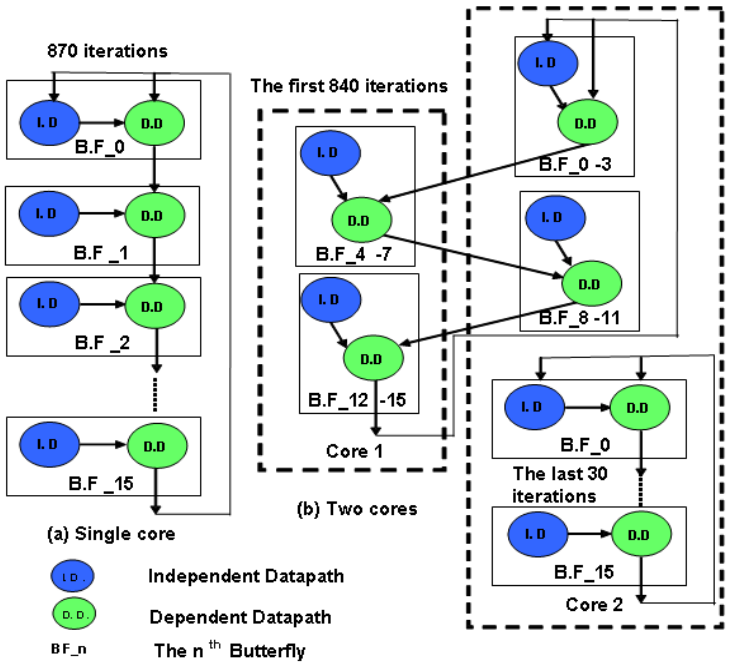
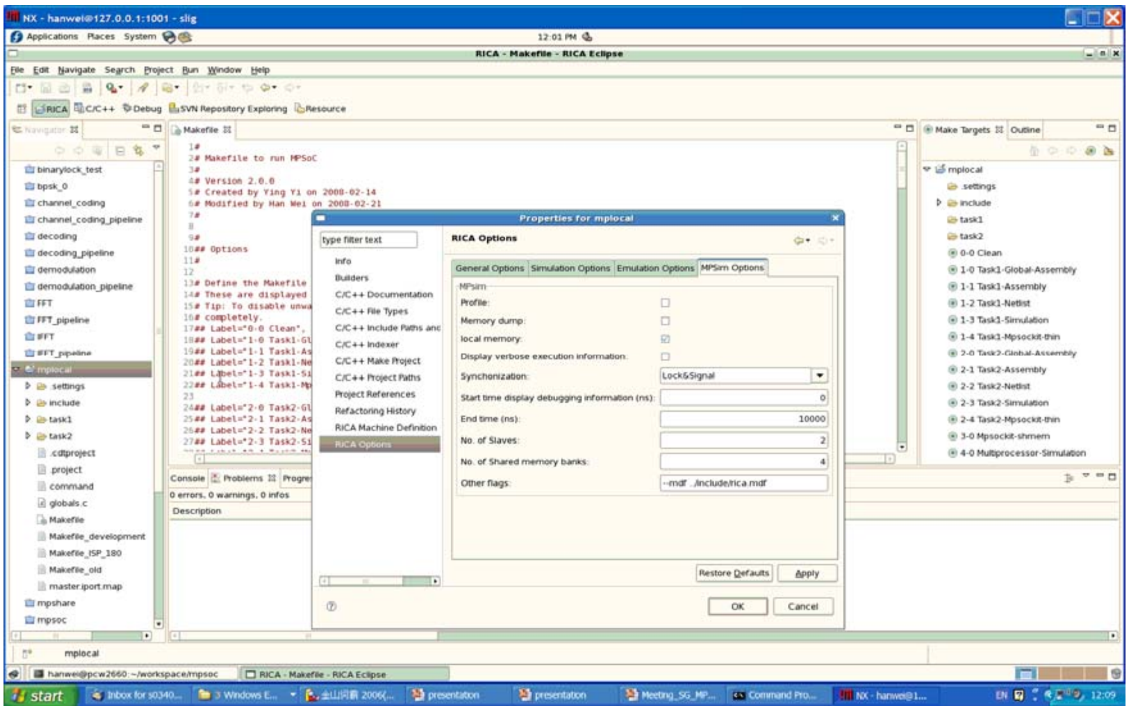


Figure 6.3 Decoding loop partitioning

iterations are assigned to the second processing core. The first core's idle time can be best absorbed in this way by executing the dual-core version of the decoding task for 840 iterations. The loop partitioning results in a balanced workload, however it may introduce a large number of data exchange between the cooperating processing cores and increase communication overheads. If the data exchange happens in the shared data memory, the shared data memory could become a performance bottleneck of the loop partitioning method due to the frequent additional data communication between the processing cores. Therefore, a feasible optimisation method is mapping the frequently read and written shared data in the dual-core version to the shared register file for reducing memory access time and memory access conflicts.

6.4 Development of multi-core projects

For easily developing a multi-core project based on the proposed mapping methodology, MRPSIM simulator has been integrated into an open source integrated development environment, Eclipse. After creating a multi-core project, the parameters for MRPSIM can



MPsim

Profile: ☐

Memory dump: ☐

local memory: ☑

Display verbose execution information: ☐

Synchonization: Lock&Signal

Start time display debugging information (ns): 0

End time (ns): 10000

No. of Slaves: 2

No. of Shared memory banks: 4

Other flags: --mdf ../include/rica.mdf

Figure 6.4 Integration of MRPSIM in Eclipse

be set through the project properties, as shown in Figure 6.4. Meanwhile, Makefiles have been written to automatically compile programs and execute them on simulators. This automation process includes compiling and executing each task on a single RICA tool flow, Mpsockit preprocessing and the execution of multiple tasks on MRPSIM simulator. Once the task partitioning method has been decided, the automation process can carry out, following the predefined settings of the tool flow in Makefiles.

6.5 Synchronisation between tasks

As mentioned in Section 4.3, two synchronisation methods, spinlock and binary semaphore, have been developed. In this thesis, there are two sync variables used for protecting a shared data item between tasks in the pipeline. One is for writing and the other is for reading. These sync variables are implemented by either spinlocks or binary semaphores. Table 6.2 shows a pseudo code for how two tasks running on two cores synchronise with each other. The writing sync variable is initialised to one, while the reading one is initialised to zero. Every time before Task 1 writes values to the shared data, it will require the writing sync variable. If this variable is one, Task 1 will set it to zero and write values to the shared data, otherwise Task 1 either continuously checks the availability of this variable or sleeps, depending on synchronisation methods. On the other side, as a concurrent task on another processing core,

Table 6.2 Synchronisation between tasks

header.h	

SYNC wlock = 1; // writing sync variable
SYNC rlock = 0; // reading sync variable
/*1: sync variable available; 0: sync variable unavailable*/
INT shared_data; // shared data item	

Task 1	Task 2

#include “header.h”
task1_function();// the main job of Task 1
require(wlock);//require writing sync variable
write(shared_data);//write values to shared data
release(rlock)//release reading sync variable
	#include “header.h”
	require(rlock);//require reading sync variable
	read(shared_data);//read values from shared data
	release(wlock);//release writing sync variable
	task2_function();// the main job of Task 2

Task 2 falls into sleep or a busy waiting state at the beginning, when it finds the reading sync variable is zero. After writing the shared data, Task 1 releases the reading sync variable by setting it to one. Meanwhile, Task 2 is aware of the availability of the reading sync variable and is able to read the shared data. After finishing reading, Task 2 releases the writing sync variable, so that Task 1 is enabled to write new values to the shared data.

6.6 Results

Following on from the discussion in Section 6.4, several homogeneous multi-core scenarios for both transmitter and receiver have been proposed. These scenarios are based on different task partitioning methods and memory architectures, as described in Tables 6.3 and 6.4. These scenarios are design examples not aiming to find a global optimum homogeneous

Table 6.3 Homogeneous multi-core scenarios for WiMAX transmitter

Scenario	No. of cores	Partitioning method	Data memory mapping
Scenario 1	2 std. cores	Task merging	Shared memory
Scenario 2	2 std. cores	Task merging	Shared & local memory
Scenario 3	2 std. cores	Loop partitioning	Shared memory
Scenario 4	2 custom cores	Loop partitioning	Shared memory & shared register file
Scenario 5	2 custom cores	Loop partitioning	Shared & local memory & shared register file
Scenario 6	8 std. cores	Task replication	Shared memory
Scenario 7	8 std. cores	Task replication	Shared & local memory

Table 6.4 Homogeneous multi-core scenarios for WiMAX receiver

Scenario	No. of cores	Partitioning method	Data memory mapping
Scenario 1	2 std. cores	Task merging	Shared memory
Scenario 2	2 std. cores	Task merging	Shared & local memory
Scenario 3	2 std. cores	Loop partitioning	Shared memory
Scenario 4	2 custom cores	Loop partitioning	Shared memory & shared register file
Scenario 5	2 custom cores	Loop partitioning	Shared & local memory & shared register file
Scenario 6	10 std. cores	Task replication	Shared memory
Scenario 7	10 std. cores	Task replication	Shared & local memory

multi-core solution for WiMAX but explore the resource/performance tradeoffs in the multi-core architecture for different system requirements. For both task merging and task replication methods, two scenarios have been designed with and without support for local data memory, respectively. These scenarios use standard cores which have the same instruction array as used for the RICA test chip. As for loop partitioning based scenarios, three scenarios have been designed with different memory architectures. For both transmitter and receiver, Scenario 3 only has shared memory used to exchange data between two processing cores, while Scenario 4 is equipped with shared register files for data exchange. Scenario 5 is an improved version of Scenario 4, with local data memory support.

Due to the use of shared register files, the cores utilised for Scenarios 4 and 5 have MULTIPBK_REG_FILE custom cells integrated. The instruction cell array configurations of both a standard core and a custom core are provided in Table 6.5. All scenarios use a

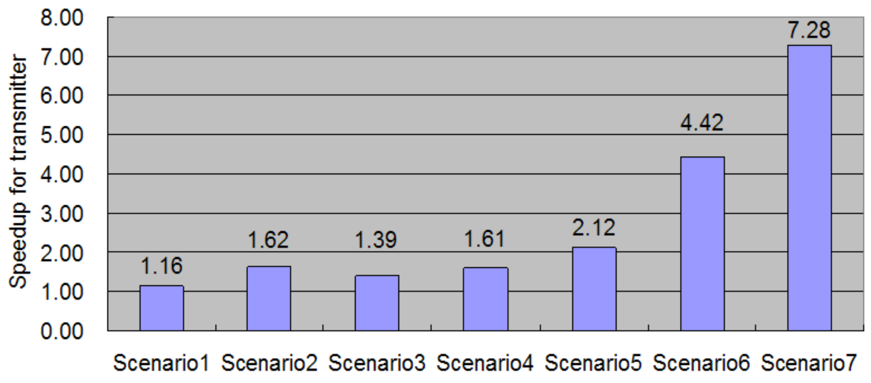
Table 6.5 Configurations of both a standard core and a custom core

Instruction Cell	Instances in Standard core	Instances in Custom core
ADD	5	5
MUL	4	4
REG	30	30
CONST	9	9
SHIFT	3	3
LOGIC	3	3
COMP	2	2
MUX	2	2
I/O REG	1	1
MEM	8	8
DMA_interface	1	1
I/O port	1	1
RRC	1	1
JUMP	1	1
MULTIPBK_REG_FILE	nil	8
Total cell number	71	79

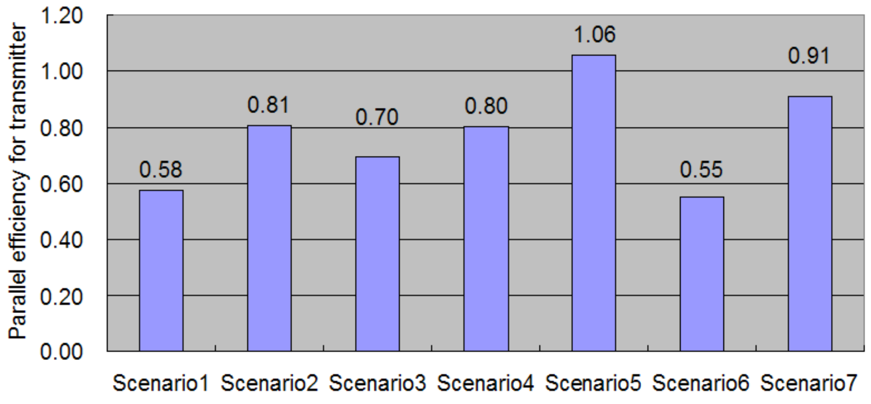
semaphore synchronisation method, since a spinlock method would cause more memory access conflicts as discussed in Section 4.3. For all scenarios, the shared data memory size is 128KB, with the access delay being 6ns. According to the memory requirement from tasks, each local memory size is 64KB and each local program memory size is 32KB. The local data memory access delay was set to 4ns. The load time from program memory differs depending on the size of each step. Only used in loop partitioning based scenarios, the shared multi-bank register file has eight banks, each of which is 32x32 bits, with the access delay being 2ns. All memory blocks have been synthesised by Faraday memory compiler Memaker using UMC 0.18μm process technology which the test RICA chip is based on. The above memory access latencies are based on generated timing results. The RRC period for each processing core was set to 2ns. Meanwhile, the transmitter and receiver were separately executed on a single RICA processor containing 128KB data memory and 128KB program memory, with 2ns RRC period and 5ns data memory access delay.

For the sake of verification, the input of the receiver is the output of the transmitter. The WiMAX PHY is based on BPSK modulation and 1/2 rate convolutional coding. To make fair performance comparisons, all scenarios are executed with 126 OFDM symbols for satisfying the replication on both transmitter and receiver. All scenarios have been implemented in ANSI C and simulated on MRPSIM simulator. Different transmitter and receiver scenarios can be combined to build various multi-core WiMAX physical layer solutions. For both transmitter and receiver, there are seven scenarios. It means that totally there could be 49 multi-core solutions based on different combinations. Figures 6.5 and 6.6 show the speedup and parallel efficiency of each scenario for WiMAX transmitter and receiver, respectively. The speedup is defined as the execution time of an application on a single-core processor divided by the execution time of the parallel version of this application on a multi-core processor. While the parallel efficiency (defined as the speedup divided by the number of processing cores) is used to estimate how efficiently processing cores are utilised in solving the problem [108].

Tables 6.6 and 6.7 provide the throughput, the average idle ratio and area of different scenarios for transmitter and receiver, respectively. The idle ratio refers to the ratio of the period when a processing core is idle to the overall simulation time. The average idle ratio is calculated by dividing the sum of the idle ratios by the number of cores. An overall multi-core processor area is estimated by accumulating the area of the instruction arrays and all memory blocks. The area figures are gained from the logic synthesis of individual RICA

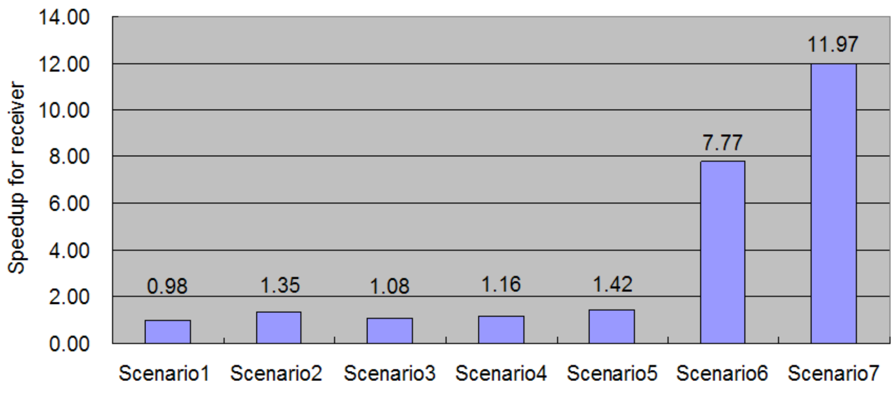


(a) Speedup

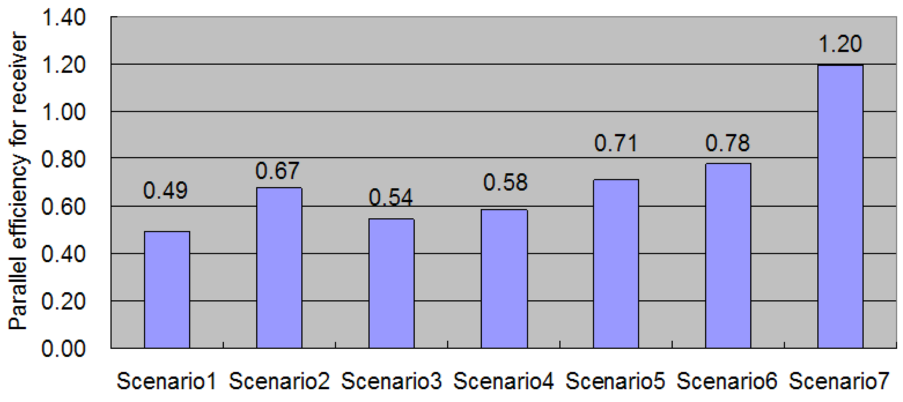


(b) Parallel efficiency

Figure 6.5 Speedup and parallel efficiency of transmitter scenarios



(a) Speedup



(b) Parallel efficiency

Figure 6.6 Speedup and parallel efficiency of receiver scenarios

cores and memory blocks based on UMC 0.18 μm process technology.

As shown in these figures and tables, in the example scenarios, task replication achieves the highest speedup compared to other task partition methods. This is mainly because task replication employs more processing cores and dispatches more balanced workload to cores. Obviously, the scenarios utilising the local data memory have better profiling in most performance metrics compared to their counterparts which have shared data memory only. This improvement is due to the data locality which reduces the accesses to the slower shared data memory. With the local data memory, several scenarios, such as Scenario 5 for transmitter and Scenario 7 for receiver, gain a super linear speedup [108] where the speedup is greater than the number of processing cores. However, the use of local data memory

Table 6.6 Performance comparison for transmitter scenarios

Scenario	Throughput (Kbps)	Average idle ratio	Area (mm^2)
Single std. core	445	-	13
Scenario 1	514	37.6%	9.96
Scenario 2	720	38.6%	15.93
Scenario 3	620	4.2%	10.03
Scenario 4	715	0.3%	10.14
Scenario 5	942	7.5%	16.11
Scenario 6	1,967	3.7%	18.91
Scenario 7	3,241	3.8%	42.78

Table 6.7 Performance comparison for receiver scenarios

Scenario	Throughput (Kbps)	Average idle ratio	Area (mm^2)
Single std. core	55	-	13
Scenario 1	54	44.1%	9.96
Scenario 2	74	43.3%	15.93
Scenario 3	60	15.2%	10.03
Scenario 4	64	14.8%	10.14
Scenario 5	78	14.3%	16.11
Scenario 6	428	2.7%	21.90
Scenario 7	659	1.4%	51.74

introduces a significant area overhead.

Due to the deeply unbalanced workload, task merging based scenarios suffer from about 30% - 40% average idle ratios and thus achieve lower throughputs. With the same number of processing cores as task merging, both transmitter and receiver Scenario 3, which use loop partitioning, have much lower average idle ratios compared to task merging based scenarios, however their performance is slightly worse. This is due to a large amount of data exchange between the two cores through the shared memory. The execution time is dramatically increased, since the loop partitioning introduces additional memory access instructions (read and write memory) and synchronised waiting time caused by memory access conflicts. However, this problem is resolved by providing a shared register file between cores, as demonstrated by Scenarios 4 and 5 which have better performance than their task merging counterparts, with a very small area overhead introduced by the shared register file. Even though the scenarios employing task replication provide the best performance, for multi-core systems which contain a limited number of processing cores, task merging and loop partitioning based solutions could be feasible alternatives.

Currently, most of certified fixed WiMAX products are working on 3.5GHz frequency band with 3.5MHz channel bandwidth, using Time Division Duplex (TDD) and 256-subcarrier

OFDM. According to Table 2.5 in [18], the fixed WiMAX profile, based on 3.5 MHz channel bandwidth and TDD 3:1 downlink-to-uplink ratio, requests data rates at 946Kbps and 326Kbps for BSPK based downlink (transmitter) and uplink (receiver), respectively. Obviously, the scenarios using task replication satisfy this requirement. It is reasonable that multi-core solutions achieve this performance with several small size RICA processing cores, each of which has less than 80 cells.

6.7 Summary

In this chapter, several homogeneous multi-core solutions, which combine different task partitioning strategies and memory architectures, have been presented for the WiMAX physical layer applications. A mapping methodology was proposed. Three different task partitioning methods have been applied and their impact on the system performance has been discussed. Simulation results have demonstrated the effectiveness of the proposed solutions in terms of speedup, parallel efficiency and throughput. Up to 7.3 and 12 speedups can be achieved by employing eight and ten dynamically reconfigurable processing cores for the WiMAX transmitter and receiver sections respectively. Meanwhile the throughputs provided by task replication based solutions satisfy the standard requirements.

However homogeneous based multi-core solutions suffer from many restrictions and deficiencies which result in throughput and area inefficiency. These deficiencies include restrictions in the memory architectural choices available and the fixed nature of the cores used in multi-core architectures. For example, in this chapter, the best solutions achieving highest throughput have significant area costs, as eight and ten processing cores used for transmitter and receiver, respectively. In the next chapter, a framework for the design of multi-core systems with heterogeneous dynamically reconfigurable processing cores will be introduced. By means of a design space exploration methodology, heterogeneous multi-core architectures can be tailored for a range of applications and provide better performance parameters.

Chapter 7
Heterogeneous Multi-core Solutions for WiMAX

7.1 Introduction

Currently, the bulk of commercial and research efforts in multi-core field focus on homogeneous multi-core processors. However on-chip homogeneity may work well for desktops and servers where this general architecture can offer a reasonable average power to a full range of applications. In embedded systems which target some special application domains like multi-media processing and wireless communication, homogeneity may not be an efficient solution. Allowing each processing core to better match its computation resources to dedicated application's needs, heterogeneous multi-core processors can provide higher power and area efficiency for certain performance requirements or significant performance advantage in an equivalent silicon area, compared to their homogeneous counterparts and high-complexity single-core processors. In Section 3.2.2.2, a detailed discussion about heterogeneous multi-core architectures has been given.

This chapter targets heterogeneous multi-core solutions for WiMAX, using RICA cores. The cellular configuration and memory size of each core can differ, depending on the choice of applications. In addition, an exploration methodology is proposed to search the design space for multi-core systems to find suitable solutions under certain system constraints, such as the number of processing cores. This design space exploration methodology aims to maximise the overall throughput while keeping the area cost at a low level. Meanwhile this algorithm involves various timing and area optimisation techniques targeting WiMAX applications. In this chapter, the target WiMAX physical layer uses 16QAM modulation and 1/2 overall code rate which requires an RS (255, 239) code, 1/2 rate convolutional coding and 2/3 puncturing pattern. Both transmitter and receiver are partitioned and mapped onto heterogeneous multi-core architectures.

This chapter is organised as follows: Section 7.2 introduces the design space exploration methodology. Sections 7.3 and 7.4 describe how WiMAX is optimised in terms of timing and area, respectively. Section 7.5 presents heterogeneous multi-core solutions and provides the results and comparisons.

7.2 Design space exploration

Existing work on design space exploration methodologies for heterogeneous multi-core processors include [55, 121, 122]. Both [121] and [55] focus on design space exploration methodologies for heterogeneous multiprocessor SoCs based on a commercial ASIP - Xtensa from Tensilica. In [121], the authors proposed an iterative exploration algorithm to select custom instructions, assign and schedule tasks on ASIPs. The work in [55] presented a heuristic to efficiently explore the design space for a pipeline based heterogeneous multiprocessor system. However, the work in [121] only considered the heterogeneity of processing elements in terms of custom instructions. This chapter investigates the customisation of both processing elements and memory architectures to best fit the target tasks. The heuristic presented in [55] only targeted a system configured in a pipelined manner with relatively simple task partitioning and mapping, while the exploration methodology introduced in this chapter involves a profiling-driven mapping methodology which can lead to complex partitioning such as loop level portioning.

Based on SUIF parallel compiler and MOVE processor tool flow, the work in [122] investigated a design space exploration methodology to find the best parallelisation of a given embedded application. The work did not consider the effect caused by changed workload balance, only a fixed context graph was presented. Also it is not a clear design trade off description that the work used only the number of adders to represent area costs. The proposed algorithm in this chapter checks the workload balance iteratively to keep each customised processing core has as an equivalent workload as possible.

The implementation of WiMAX on a heterogeneous multi-core architecture mainly involves task partitioning, task scheduling, task mapping, optimisation for each core and balancing the workloads among different cores. Since each core may differ, some performance metrics used in Chapter 6 (e.g. parallel efficiency) are not suitable for estimating how efficiently a heterogeneous multi-core system is utilised in solving the problem. Instead, this chapter uses the ratio of throughput to area to evaluate the efficiency of a multi-core architecture. This ratio represents a ratio of a multiprocessor performance to its hardware cost and is widely used in many research work such as [123]. In this thesis, this ratio is defined as

$$\Delta = Throughput / Total\ Area \tag{7.1}$$

For WiMAX, the ratio is separately computed for transmitter and receiver. The throughput can be obtained by dividing the bit number of one symbol input or output by the execution time for completing one symbol transmission or reception. The total area of a transmitter or a receiver indicates the area sum of shared memory blocks plus RICA cores. Each RICA core area is calculated via the function $A_j(p_i)\,(1 \le i \le I)$ where I is the number of cores supported by design space j, while p_i represents i th RICA core. This function works by accumulating the area of all instruction cells, interconnection and local data and program memory in each RICA core.

Given a characterised application (e.g. WiMAX) and constraints such as the number of cores and whether the cores support custom instruction extensions (i.e. domain-specific instruction cells), the design space exploration objective is to maximise the throughput and the throughput to area ratio (Δ) if the throughput can not be further improved. An algorithm is proposed for this optimisation problem. The pseudo code of this algorithm is shown in Table 7.1. This algorithm supports both single-core processor designs and multi-core processor

Table 7.1 Pseudo code of the design space exploration methodology

```
For Design $j$ with $I$ processing cores
  Timing optimisation on a single RICA core;
    If the instruction cell array tailorable
      Customise the instruction cell array;
    If custom instructions enabled
      Add custom instructions (e.g. Galois Field multiplier);
  Do {
     If multi-core design
       Task partitioning, scheduling and mapping;
       For each RICA core $p_i$
         Timing optimisation;
           If local data memory enabled
             Add local data memory;
       End for
  } while ( multi-core design && workloads can be further balanced)

  Do {
     For each RICA core $p_i$
       Area optimisation;
         If the instruction array tailorable
           Tailor the instruction cell array;
         If local data memory enabled
           Customise local data memory size;
     End for
  } while (multi-core design && workloads can be further balanced)

  Calculate the total area;
    Total Area = $\sum_{i=1}^{I} A_j(p_i)$ + the area of shared memory blocks;
  Calculate the overall throughput achieved for Design $j$
  $\Delta_j$ = Throughput/Total Area;
```

designs of either homogeneity or heterogeneity. In addition, this algorithm allows both standard RICA and custom RICA based designs.

For example, for a design space with I cores, firstly an optimisation of the application performance is carried out on a single RICA processor. This optimisation may be twofold. One is optimising the application code to maximally get benefits from the RICA architecture, while the other is architectural, including customised cellular configuration of the RICA core and custom instruction extensions to best fit the application. The second type of timing optimisation techniques only can be applied when the RICA instruction cell array is tailorable and extensible. The timing optimisation is detailed in Section 7.3. Then the algorithm checks whether the design is a multi-core based implementation. If so, the application (WiMAX in this thesis) will be partitioned and mapped onto multiple cores by means of the profiling-driven mapping methodology introduced in Section 6.2. This methodology may generate a partitioning and mapping result based on task merging, task replication, loop partitioning or their combination. After that, tasks are separately optimised for reducing the execution time on the processing cores which they are assigned to. This optimisation is similar as that performed on a single RICA processor, except that a local data memory can be added for each core during this optimisation. Then if the design is multi-core based, the workload balance will be checked in terms of the execution time of tasks on each core and the idle ratio of each cores. For example, if one core has 10% idle ratio and the other has 40% idle ratio, the workload is not balanced. If the workloads of the cores are not balanced, the procedure will return to the step of task partitioning and mapping to generate a more balanced solution and restart the timing optimisation. If a better workload balance can not be achieved due to the limitation of the code itself, the procedure will switch to do the area optimisation for each core, which is discussed in Section 7.4. Since the area optimisation may break the workload balance, there is another workload balance check performed after it. Then the procedure either starts a new iteration for the area optimisation or breaks the loop. Finally, the total area, throughput, and thus Δ are calculated. Through the efficient task partitioning and mapping, timing and area optimisation as well as workload balance check, this algorithm tries to maximise throughput and/or Δ for each given design space which may have a variety of implementations due to different configurations of RICA cores.

In this chapter, both WiMAX transmitter and receiver are implemented on heterogeneous multi-core architectures by using this design space exploration methodology. Figures 7.1 (a) and (b) show the main components of 16QAM based WiMAX transmitter and receiver, respectively. The arrows indicate the data flow through the various function components. As mentioned in Section 6.3.1, the transmitter can be functionally split into two main parts, channel coding and OFDM downlink processing. The next two sections discuss the timing and area optimisation techniques employed by this design space exploration methodology. Some of these optimisation techniques are application specific, but many of them are general enough and work for different range of applications. Currently, this exploration methodology is performed manually. An automatic design space exploration would be a main focus in the future work.

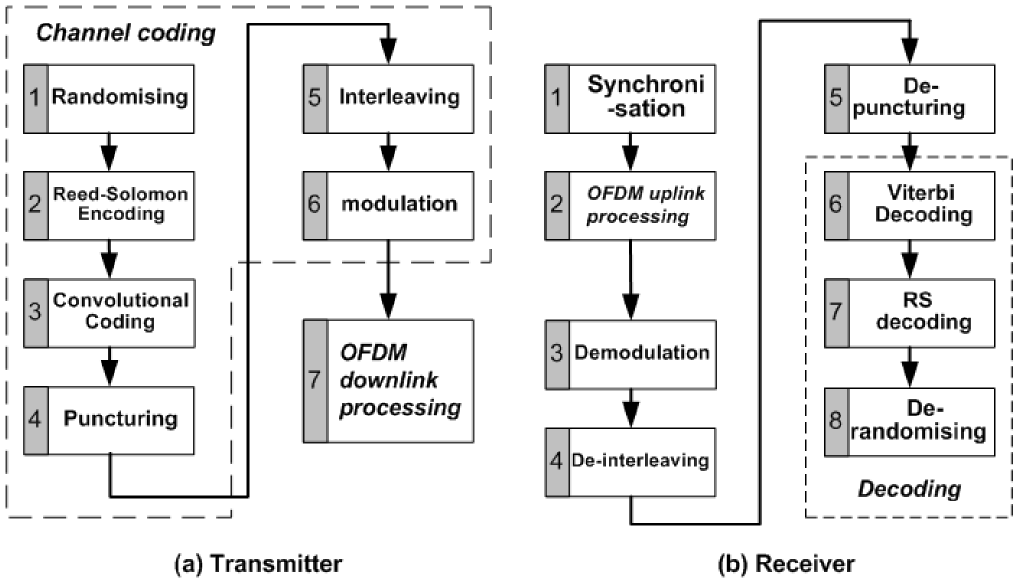


Figure 7.1 The main blocks in 16QAM based WiMAX transmitter and receiver

7.3 Timing optimisation

7.3.1 Code optimisation

As mentioned in the last section, timing optimisation involves two categories of optimisation techniques. The first one targets optimisation of the code to fit the characteristics of the RICA architecture better. The RICA architecture is designed to support only one jump operation per step. Shown in the left-hand side of Table 7.2, the original code generates two jumps and involves three steps. Therefore, it is more efficient to implement these small branches by multiplexers instead of conditional statements. Table 7.2 also gives the optimised code which can fit into a single step and make better use of the available resources. Moreover, with the optimised code, the instruction cell array does not need to be reconfigured at all and hence is free of reconfiguration overhead. More benefits can be gained from this technique, if such a piece of code is a self-loop. For WiMAX physical layer, this replacement technique is mainly used in functions such as demodulation and pilot insertion which include many branches.

Table 7.2 A code example of replacing jumps with multiplexers

Original Code	Optimised Code

ctrl = input_1 * input_2;
if (ctrl > t){
  output = (input_1 + input_2) * (input_1 – input_2);
}
else{
  output = input_1 – input_2;
}
	ctrl = input_1 * input_2;
	result_0 = (input_1 + input_2) * (input_1 – input_2);
	result_1 = input_1 – input_2;
	
	output = (ctrl > t) ? result_0 : result_1;

Duo to the long memory access latency and the limited memory access interface cells, it is very expensive to access memory. Another code optimisation technique is reducing memory access count by storing constants in registers. For example, 16QAM modulation has 16 constellation points. As shown in Figure 2.6, the value of the real part or imaginary part of

Table 7.3 Storing constants in registers for 16QAM modulation

Original code	Optimised code

//store fix point representations in memory
const int real[16] ={ 324, 324, 324, 324, 971, 971, 971, 971, -324, -324, -324, -324, -971, -971, -971, -971};
 	
const int imaginary[16] ={324, 971, -324, -971, 324, 971, -324, -971, 324, 971, -324, -971, 324, 971, -324, -971};
 	
for (i = 0, j=0; i< Ncbps; i+=4, j++)
//Ncbps: coded bits per symbol
{
  addr = addressCalculation();
  real_out[j] = real[addr];
  imag_out[j] = imaginary[addr];
}
	for (i = 0, j=0; i< Ncbps; i+=4, j++)
	{
	  addr = addressCalculation();
	  pos = (addr < 4) ? 324:971;
	  neg = (addr < 12) ? -324:-971;
	  real_out[j] = (addr < 8) ? pos : neg;
	  ctrl1 = addr & 1;
	  pos = (ctrl1 == 0) ? 324:971;
	  neg = (ctrl1 == 0) ? -324:-971;
	  ctrl2 = addr & 2;
	  imag_aout[j] = (ctrl2 == 0) ? pos : neg;
	}

each constellation point is one of four figures (i.e. $3/\sqrt{10}$, $1/\sqrt{10}$, $-1/\sqrt{10}$ and $-3/\sqrt{10}$). Therefore it is feasible to store these constants into registers instead of memory. Table 7.3 shows the original and optimised codes for 16QAM modulation. In the left-hand side, the fix point representations of these values are stored in memory. The loop body requires two write memory and six read memory operations, four of the read accesses from the function of addressCalculation. RICA processor supports no more than four write and read memory accesses within one single step, therefore this loop body can not be placed in one step. In the right-hand side code, a bit little complex calculation is carried out by using logic operations and multiplexers. The number of read accesses is reduced to four, so that this loop body can be executed within one step. This optimisation results in 48.3% execution time saving for 16QAM modulation.

If a self-loop iterates many times and the loop body can fit into one single step, it is worth using software pipelining to reorganise the loop by inserting registers in long data-path chains. As a result, the original critical paths can be divided into several portions executing in parallel within one single step. In RICA implementations, software pipelining is supported by means of inserting a compiler detective into the loop body. However software pipelining involves additional registers and corresponding routing resources. Meanwhile, a software pipelined step is split into several logical pipeline stages which require extra steps for the prologue and epilogue. Hence software pipelining is not suitable for a loop without a large number of iterations. For WiMAX applications, software pipelining is used in functions like pilot insertion and convolutional coding. For example, by using software pipelining, the convolutional coding function can have 28.6% throughput improvement, while the optimised 16QAM modulation code in Table 7.3 can have 53.6% further reduction in execution time.

Another optimisation technique is common subexpression elimination which replaces the identical subexpression in multiple equations with a variable holding the computed value. Hence the number of operations can be reduced. This technique is employed in functions such as convolutional coding where the two generator polynomials have a common subexpression $1+X^2+X^3+X^6$. Other optimisation techniques including loop unrolling and loop splitting have been used in functions like RS coding. The main idea behind these code optimisation techniques is to make maximal use of available resources and reduce the step number, especially for self-loops.

7.3.2 Architectural optimisation

In contrast to maximally utilising resources within each step for a given number of instruction cells, the second type of optimisation is architectural. It involves tailoring and extending the instruction cell array to best fit the dedicated program. For example, the optimised code in Table 7.2 could not be placed in one step, if the instruction cell array only has one MUL cell. By carefully increasing required resources to satisfy the requirement of such a piece of code, especially those self-looping code, the reconfiguration overhead for the whole application can be dramatically reduced. Moreover, extending the instruction cell array by adding user-defined custom instruction cells can speed up functions and allow smaller, more cost efficient designs. In this chapter, a custom instruction cell called GFMULT [30] is used to implement Galois Finite Field multiplication which is the most time-consuming operation in RS encoding/decoding and costs much more silicon area and execution time if implemented by standard instruction cells. The GFMULT cell is implemented by the custom cell generation environment introduced in Section 4.4. In addition, other custom instructions such as MULTIPTBK_REG_FILE introduced in Section 4.2.2.2 can be used to support more memory architectural choice. In Chapter 3, Table 3.3 lists both standard and custom instruction cells. For standard instruction cells from I/O REG to JUMP, the instance numbers are fixed due to the architectural characteristics of RICA, while the instance numbers of other cells are allowed to be changed through altering their values in MDF. Another architectural optimisation is adding a local multi-bank data memory for each processing core. By equipping local data memory, the conflict in accessing the shared memory is alleviated, and thus the throughput is improved. The MRPSIM simulator can provide information of the local data memory requirement for each processing core according to allocated tasks. Thereby the local data memory sizes for individual cores are not necessarily the same and can be customised to meet the minimum memory demands, so that the memory access can be accelerated.

7.4 Area optimisation

This design space exploration methodology employs varied area optimisation techniques for different situations. When the workload is balanced, the area optimisation just removes those redundant instruction cells existing in processing cores. It will introduce insignificant influence on the execution time and hence still keep the balance between cores. For example, there are no multiplication operations required by those functions in channel coding using the optimised code. Therefore it is unnecessary to keep costly MUL cells in a processing core customised for performing channel coding only. The second type of optimisation technique is applied to the situation where the workload is unbalanced, even after greatest efforts have been paid to task partitioning and mapping as well as timing optimisation.

Basically the throughput of a system is determined by the most time-consuming task. Even though other tasks perform much better, the overall throughput would not be improved. This unbalance results in those processing cores with light workloads to spend a considerable time in idle on waiting for the inter-processor synchronisation. For example, in a transmitter solution with only two processing cores, after timing optimisation and merging all functions of channel coding into a single task, task OFDM downlink processing still takes much more execution time than the merged task. Therefore, task OFDM downlink processing determines the transmitter throughput which the merged task can not make further contributions to improve any more. In addition, partitioning methods like task replication can not be employed to such a system due to the limited number of processing cores. For this case, the area optimisation tries to bring a workload balance between cores through cutting down the resources for those highly underloaded cores (e.g. the core running the merged task) until all tasks have a close execution time or balanced workloads. Although it will not affect the throughput of applications too much even after considering the memory competition and inter-processor synchronisation, the ratio of throughput to area will be improved with the area reduction. In all cases, area optimisation techniques intend to maximise the area efficiency without breaking the workload balance or worsening the throughput. Moreover, both local data and local program memory sizes can be tuned to meet the minimal requirements from tasks assigned to each core, through modifying the multi-core MDF configuration.

7.5 Results

A group of single-core or multi-core processor designs were developed for transmitter and receiver, respectively. These designs have been configured by different constraints as examples for the design space exploration. These constraints include the number of processing cores, tailorability and extensibility. Each design may have a large design space in terms of the instruction cell array configuration of each individual core, local memory size, mapping methods and so on. A good solution can be found for each design by proper partitioning and mapping as well as timing and area optimisation.

Tables 7.4 and 7.5 show the solutions found by means of the design space exploration methodology for each transmitter and receiver design, respectively. The second columns of the two tables show the number and types of cores used in each design. For both transmitter and receiver, there are two single-core based designs, while others are multi-core designs, all of which own local data memory. Design 2 of transmitter is developed as a homogeneous multi-core solution which employs two standard RICA cores with equal local memory size. Designs 4-6 of transmitter and Designs 3-6 of receiver are heterogeneous based solutions where the number of processing cores gradually increases. According to the task mapping and scheduling results, the exploration stops at the quad-core and five-core solutions for transmitter and receiver, respectively. The third column in both of the two tables indicates

Table 7.4 Comparison of transmitter design configurations

Design	No. and types of cores	Tailorable& Extensible	Mapping	Local data memory (KB)	Local program memory (KB)
Design 1	1 standard core	No	1-7	128	128
Design 2	2 standard cores	No	(1-2); (3-7)	48;48	128;128
Design 3	1 RICA1	Yes	1-7	128	128
Design 4	1 RICA2 & 1 RICA3	Yes	(1-6); (7)	48;8	128;8
Design 5	1 RICA4 & 2 RICA3	Yes	(1-6); (7)	48;8	128;8
Design 6	1 RICA5 & 3 RICA3	Yes	(1-6); (7)	48;8	128;8

Table 7.5 Comparison of receiver design configurations

Design	No. and types of cores	Tailorable& Extensible	Mapping	Local data memory (KB)	Local program memory (KB)
Design 1	1 standard core	No	1-8	128	128
Design 2	1 RICA6	Yes	1-8	128	128
Design 3	1 RICA7 & 1 RICA6	Yes	(1-5); (6-8)	16;16	16;64
Design 4	1 RICA8 & 2 RICA9	Yes	(1-5,7,8);(6)	16;16	64;8
Design 5	1 RICA10 & 3 RICA6	Yes	(1-5); (6-8)	16;16	16;64
Design 6	1 RICA11 & 4 RICA6	Yes	(1-5); (6-8)	16;16	16;64

whether the design is allowed to be tailored and extended.

The RICA cores used in transmitter Designs 3-6 and receiver Designs 2-6 are allowed to be tailored and custom instruction extended. Including the standard RICA core and its custom variants, totally twelve types of RICA core configurations have been obtained for these solutions through the design space exploration. Tables 7.6 and 7.7 provide the specification of each core type used for transmitter and receiver, respectively. The specification includes the instance number of each instruction cell, local memory size, area cost as well as the tasks each core is customised for. The area figures are obtained from the RICA multiple product

Table 7.6 Comparison between standard and custom RICA cores used in transmitter designs

Instruction cell	Standard	RICA1	RICA2	RICA3	RICA 4	RICA 5
ADD	5	9	1	5	7	7
MUL	4	2	nil	2	nil	nil
REG	30	69	24	28	89	89
CONST	9	35	9	9	32	32
SHIFT	3	19	2	3	4	19
LOGIC	3	22	3	1	22	22
COMP	2	13	2	2	13	13
MUX	2	7	2	1	7	7
I/O REG	1	1	1	1	1	1
MEM	8	8	8	8	8	8
DMA_interface	1	1	1	1	1	1
I/O port	1	1	1	1	1	1
RRC	1	1	1	1	1	1
JUMP	1	1	1	1	1	1
GFMULT	nil	4	4	nil	4	4
Total cell number	71	196	60	64	191	206
Local data memory (KB)	128/48	128	48	8	48	48
Local program memory (KB)	128	128	128	8	128	128
Core area (mm^2)	12.94/9.62	13.86	9.21	2.17	10.02	10.25
Targeting tasks	General	Transmitter	Channel coding	OFDM	Channel coding	Channel coding

Table 7.7 Comparison between standard and custom RICA cores used in receiver designs

Instruction cell	Standard	RICA6	RICA7	RICA8	RICA9	RICA10	RICA11
ADD	5	16	1	13	11	3	13
MUL	4	2	1	4	nil	2	4
REG	30	69	14	46	36	28	31
CONST	9	26	6	15	20	6	15
SHIFT	3	7	1	4	6	2	4
LOGIC	3	25	1	8	12	1	2
COMP	2	26	1	26	12	2	3
MUX	2	19	1	19	11	2	6
I/O REG	1	1	1	1	1	1	1
MEM	8	8	8	8	8	8	8
DMA_interface	1	8	1	1	1	1	1
I/O port	1	1	1	1	1	1	1
RRC	1	1	1	1	1	1	1
JUMP	1	1	1	1	1	1	1
GFMULT	nil	1	nil	8	nil	nil	nil
Total cell number	71	213	39	156	121	59	91
Local data memory (KB)	128	128/16	16	16	16	16	16
Local program memory (KB)	128	128/64	16	64	8	16	16
Core area (mm^2)	12.94	14.07/6.69	2.87	4.56	2.93	3.11	3.53
Targeting tasks	General	Receiver	Synch, OFDM, etc.	Receiver except Viterbi	Viterbi	Synch, OFDM, etc.	Synch, OFDM, etc.

wafer test chip data sheet and memory logic synthesis.

The mapping solutions for each design are given in the fourth columns of Tables 7.4 and 7.5. For instance, in transmitter Design 5, tasks 1 - 6 shown in Figure 7.1(a) are assigned to one RICA4 core while Task 7 is replicated on two RICA3 cores. All multi-core solutions have the same amount of shared memory resources – 128K bytes with 6ns access delay, and each core may have different sizes of local data and program memory as shown in the last two columns of Tables 7.4 and 7.5. For example, in transmitter Design 5, RICA4 has a local data memory sized 48KB and 128KB program memory, while both local data memory and program memory for each RICA3 are 8KB. The access delays for local memory differ depending on the size of the local memory. The RRC period for all designs was set to 2ns.

Tables 7.8 and 7.9 show the results for each design of transmitter and receiver, respectively, including throughput, area, idle ratio and the ratio of throughput to area. The areas are calculated by the equation described in Table 7.1. All throughput and idle ratio figures are provided by MRPSIM simulator. For transmitter designs, it is clear that based on standard RICA processing cores, both Design 1 and Design 2 deliver very poor performance. Even equipped with homogeneous dual-core and faster local data memory, Design 2 exhibits only 12% improvement in performance, but introduces much more area overheads compared to Design 1. This is mainly due to the unbalanced workload which results in a very high

Table 7.8 Results from the exploration methodology for transmitter designs

Design	Throughput (Kbps)	Area (mm^2)	Average idle ratio	Throughput/Area (Kbps per mm^2)
Design 1	21.7	12.94	nil	1.68
Design 2	24.36	25.21	49.5%	0.97
Design 3	2,094.14	13.86	nil	154.43
Design 4	2,910.93	17.35	1.5%	167.82
Design 5	5,805.51	18.94	4.9%	306.46
Design 6	8,669.47	19.96	7.3%	434.29

Table 7.9 Results from the exploration methodology for receiver designs

Design	Throughput (Kbps)	Area (mm^2)	Average idle ratio	Throughput/Area (Kbps per mm^2)
Design 1	3.99	12.94	nil	0.31
Design 2	476.60	14.07	nil	33.88
Design 3	810.22	15.54	13.3%	52.15
Design 4	1,313,68	18.14	26.4%	72.40
Design 5	1,868.27	29.16	1.2%	64.07
Design 6	2,430.66	36.28	3.8%	67.01

average idle ratio of Design 2. The reason for this unbalance is that one processing core executes a very heavy computation for RS encoding, while the other is almost idle at all time because the execution time consumed by the rest of WiMAX transmitter functions on it is negligible, compared to RS encoding.

When the instruction cell array of a processing core is allowed to be tailored as well as extended by domain-specific custom instruction cells (in this case, GF multiplier cells), much better performance is achieved as can be seen in Design 3. Benefiting from a heterogeneous dual-core implementation, Design 4 improves over Design 3, and has about 30% area reduction compared to its homogeneous counterpart Design 2. In both Design 5 and Design 6, Task OFDM downlink processing is replicated on multiple RICA3 cores, while the other processing core runs all other functions. Hence the OFDM downlink processing task can be executed in parallel for multiple consecutive OFDM symbols. Although owning only one more RICA core, Design 6 has much better performance than Design 5, and achieves the highest throughput and throughput to area ratio among all transmitter designs. This is because the execution time of the optimised OFDM downlink processing is nearly three times that of the optimised channel coding. In Designs 4 and 5, the numbers of cores are limited to two or three. Hence both Designs 4 and 5 utilise the second type of area optimisation technique, introduced in Section 7.4, which brings these designs low area costs and average idle ratios without hurting throughputs. As a result, RICA2 and RICA4 are generated as small versions of RICA5 which has more resources for the channel coding task. Compared to RICA5, RICA2 and RICA4 have 64% and 15% area reductions in terms of the instruction cell array including cells and interconnection, respectively. It results in 10.2% and 2.3% savings for the overall processing core area, as can be seen in Table 7.6.

For the receiver side, same as the transmitter, the custom instruction cell GF multiplier brings a significant improvement for both throughput and area cost for those designs allowed to be extended. Due to the fact that the execution time of Viterbi task takes about 70% of the overall time of a receiver, in Designs 3, 5 and 6, the receiver is partitioned into two batches of tasks. Tasks 1-5 are allocated to one RICA core, while tasks 6-8 are allocated to one RICA6 core or replicated to up to four RICA6 cores. Since the execution time proportion between tasks 1-5 and tasks 6-8 is almost one to four, Design 6 demonstrated much better results for throughput and the ratio of throughput to area, compared to Designs 3 and 5. In Designs 3 and 5, RICA7 and RICA10 are optimised by means of the second type of area optimisation technique. Compared to RICA11, RICA7 and RICA10 are 56% and 36% smaller in the instruction cell array area, respectively, in the meantime 19% and 12% smaller in the overall processing core area, respectively.

Different from the previous mapping and scheduling, Design 4 uses a mapping and scheduling method where the most time-consuming task Viterbi is replicated on two RICA9 cores and all other tasks are assigned to one RICA8 core. This mapping and scheduling is illustrated in Figure 7.2 where the processing of four OFDM symbols is demonstrated. The left-hand side of Figure 7.2 shows the receiver task graph, while the percentages represent the proportion of each task batch execution time in the overall execution time. On RICA8, first of all tasks 1-5 execute four consecutive symbols, and then tasks 7-8 perform four consecutive symbols once task 6 finishes the corresponding symbols on two RICA9 cores. This scheduling can achieve better workload balance than the normal task replication for the triple-core design. Design 4 does not employ the second type of area optimisation technique, but achieves the highest throughput to area ratio in all receiver designs. Due to the computational complexity of the receiver, both throughputs and the ratios of throughput to area in receiver designs are much lower than those for transmitter designs. According to

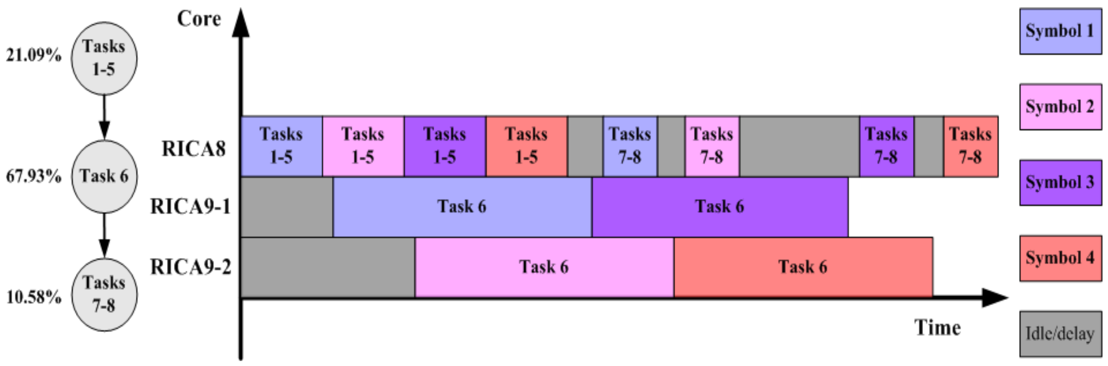


Figure 7.2 Task partitioning, mapping and scheduling in receiver Design 4

Table 2.5 in [18], the fixed WiMAX profile, which uses 3.5 MHz channel bandwidth and TDD 3:1 downlink-to-uplink ratio, requests data rates at 3,763Kbps and 1,306Kbps for downlink and uplink based on 16QAM modulation with 1/2 code rate, respectively. It is clearly shown in Tables 7.8 and 7.9 that Designs 5 and 6 of transmitter as well as Designs 4-6 of receiver satisfy these requirements.

In Table 7.10, the design with the best throughput (i.e. the combination of Design 6 for both transmitter and receiver) is compared with combined Scenario 7 (i.e. the combination of Scenario 7 for both transmitter and receiver) proposed in Chapter 6 and several other WiMAX solutions based on commercial high performance multi-core processors. In Table 7.10, for the sake of direct comparison, the throughput is the sum of transmitter and receiver throughputs. In addition, throughputs are estimated at 90nm technology basis for those solutions based on other process technologies. For 180 nm and 130 nm technologies, the estimated throughputs are double and 1.5 times as the actual throughputs, respectively. Shown in Table 7.10, Design 6 offers a much higher throughput than Scenario 7 in Chapter

Table 7.10 Comparison of multi-core solutions

Multi-core solution	Structure	Process technology (nm)	Frequency (MHz)	Throughput (estimated for 90mn) (Mbps)
Design 6	9 RICA custom cores	180	500	22
Scenario 7 in Chapter 6	18 RICA std. cores	180	500	7.8
PC7218 [94]	2 PC102 (2 x 308 cores)	130	160	40
SB3010 [61]	One ARM9 and 4 Sandblaster cores	90	600	2.9
IXP2350 [59]	one XScale core and 4 microengines	90	1,200 (XScale)/ 900 (microengine)	25
MSC8126 [26]	4 SC104 cores plus Viterbi coprocessor	90	500	33.7
CELL [97]	one 64-bit PPE and eight SPEs	90	3,200	40

6 at almost half area cost. It proves that even with more workloads (e.g. RS coding), heterogeneous multi-core designs surpass homogeneous multi-core designs in performance with much less silicon costs. In addition, the performance of both Design 6 and Scenario 7 deliver good performance, even though the proposed solutions contain a few simple RICA cores operating at not very high frequency and are currently based on a test chip with a process technology which is one or two generation behind other solutions. One reason behind this is that the RICA architecture's salient characteristics (e.g. customisable instruction cell array and reconfigurable rate control) bring the proposed multi-core solutions advantages over DSP based multi-core solutions. It is also partially because all processing cores in both Design 6 and Scenario 7 have very low average idle ratios, compared to other solutions. For example, in [97] only five of SPEs in CELL are used for processing WiMAX, and the SC 104 cores in Freescale MSC8126 have only an average loading rate of 70% [16]. Moreover, combined Design 6 and Scenario 7 have area costs at 55.75 mm^2 and 94.53 mm^2, respectively. However, some of other multi-core solutions take up more space, for example, the die size of a CELL processor is 221 mm^2 at 90nm process technology.

7.6 Summary

In this chapter, a design space exploration methodology has been proposed to find good design solutions under certain system constraints. This algorithm is suitable for both single-core based and multi-core based designs including heterogeneous and homogeneous multi-core solutions. The ratio of throughput to area was used for estimating the efficiency of a multi-core design. In addition, a variety of timing and area optimisation techniques were introduced and used within this algorithm to improve the throughput as well as area efficiency and thus the throughput to area ratio. Several designs have been developed and good improvement and trade off have been obtained through the exploration methodology for each design. Totally eleven types of custom RICA cores have been devised for these designs. Results demonstrated that heterogeneous multi-core architectures can provide throughputs of up to 8.7 Mbps and 2.4 Mbps for transmitter and receiver, respectively, meanwhile achieving a ratio of throughput to area up to 200 Kbps per mm^2 for the overall WiMAX physical layer. In addition, a comparison with other WiMAX multi-core solutions was provided to demonstrate that the best solution Design 6 delivers a very good throughput at relatively low area cost.

Chapter 8
Multitasking WiMAX on an RTOS

8.1 Introduction

Nowadays, complicated embedded system designs are driven by the increasing demand for high performance real-time applications. The approach of software applications plus bare machines, which may work well for microwave ovens, can not work for consumer electronics where multiple tasks even multiple processors are involved. As the RICA architecture targets on embedded systems, it is very important to have a real-time operating system ported to RICA for satisfying the multitasking requirement. Moreover, for embedded systems targeting deterministic applications (e.g. WiMAX), it is worth to have an RTOS to maintain the system reliability/stability, protect the system from a crash caused by some faults, and diagnose the faults when a crash happens. From multi-core systems point of view, an operating system allows for dynamic scheduling which can schedule tasks at run-time. As shown in Figure 4.1, an OS can be ported to the master core which takes charge of task management and scheduling, while all slave cores act as I/O processors. Meanwhile, some OSes have a load balancer or support task stealing which can automatically balance workloads among processing cores regarding run-time situations. In addition, an OS enables multithreading which can make better advantage of computing resources. As multithreading on multi-core systems is one of main future targets for the work originated from this thesis, porting an RTOS on a single RICA processor is the first compulsory step for this target.

An RTOS or embedded operating system is a type of OS specially designed for real-time applications, including those running on consumer electronics, industrial robots, industrial control, and scientific research equipment. The major difference between an RTOS and a normal desktop/server OS is that an RTOS must satisfy the strict deadline of real time systems. RTOSes can be classified into hard real-time RTOSes and soft real-time RTOSes. For the first category of RTOS, such as those used in automobiles and health care, the deadline must be deterministically met. Otherwise applications would fail and cause serious accidents which may even threaten users' lives. However the effort of satisfying the deadline is somewhat relaxed in soft real-time RTOSes, such as those used in customer electronics. In this kind of embedded system, the operations completed after the deadline are still meaningful, for example the delayed live video. Usually an RTOS is devised to be very compact and efficient, due to the limited resources of embedded systems. The minimal interrupt and thread switching latencies are often used to estimate how efficient an RTOS is. Currently, there are many RTOSes available for various hardware platforms. As for reconfigurable computing architectures, work so far has included mapping an RTOS on soft-core processors, such as Altera NiosII [124] and Xilinx MicroBlaze [125], which have been synthesised on fine-grained FPGAs. However, the overhead introduced by the communication between the underlying logic blocks has an adverse effect on power consumption and area.

This chapter focuses on porting an RTOS – Micro C/OS-II to the RICA architecture and the multitasking implementation of a WiMAX physical layer program on a single RICA with this RTOS support. For comparison's sake, the WiMAX program has been also implemented on the ARM7TDMI processor with Micro C/OS-II support. In Section 8.2, a survey is given for choosing a suitable RTOS. Section 8.3 describes both hardware and software requirements for developing this porting. Section 8.4 details how the multitasking WiMAX program works on Micro C/OS-II. Finally, in Section 8.5, the results are given to demonstrate the overheads introduced by this RTOS and compare the performance of multitasking based WiMAX on RICA and ARM7TDMI.

8.2 The selection of RTOSes

To choose a suitable RTOS for exploring multitasking on RICA, thirty-three RTOSes have been investigated from both the open source community and proprietary sources. All of selected RTOSes can be categorised in term of the application domain, source type, kernel type and the derivation shown in Table 8.1.

Table 8.1 Survey of RTOSes

RTOS	Application	Source type	Kernel type	Derivation
eCos [126]	General	Open source	Microkernel	N/A
Fiasco [127]	General	Open source	Microkernel	N/A
FreeRTOS [128]	General	Open source	Microkernel	N/A
NetBSD [129]	General	Open source	Monolithic	BSD
Phoenix-RTOS [130]	General	Open source	Microkernel	Unix-like
QNX [131]	General	Open source	Microkernel	Unix-like
RTEMS [132]	General	Open source	Microkernel	N/A
ThreadX [133]	General	Closed source	Picokernel	N/A
TRON [134]	General	Open source	N/A	N/A
uCLinux [135]	General	Open source	Monolithic	Linux
VxWorks [136]	General	Closed source	N/A	N/A
Xpe [137]	General	Closed source	Hybrid kernel	Windows XP
MontaVista Linux [138]	Handheld devices	Open source	Monolithic	Linux
Nucleus RTOS [139]	Handheld devices	Closed source	N/A	N/A
Palm OS [140]	Handheld devices	Closed source	N/A	N/A
Prex [141]	Handheld devices	Open source	Microkernel	N/A

Symbian OS [142]	Handheld devices	Closed source	Microkernel	N/A
Windows Mobile [143]	Handheld devices	Closed source	N/A	WinCE
WinCE [144]	Handheld/ industrial devices	Shared source	Monolithic	N/A
Integrity [145]	Hard real-time	Closed source	Microkernel	N/A
LynxOS [146]	Hard real time	Closed source	N/A	Unix-like
Micro C/OS-II [147]	Hard real time	Open source	Microkernel	N/A
RTLinux [148]	Hard real-time	Open source	Monolithic	Linux
Agnix [149]	Education	Open source	Monolithic	Linux
Minix [107]	Education	Open source	Microkernel	Unix-like
ChorusOS [150]	Network	Open source	Microkernel	N/A
Contiki [151]	Network sensors	Open source	N/A	N/A
ETLinux [152]	Industrial computers	Open source	Monolithic	Linux
Freesco [153]	Routers	Open source	Monolithic	Linux
Inferno [154]	Distributed services	Open source	Monolithic	Unix
Nut/OS [155]	Ethernet	Open source	N/A	N/A
picoOS [156]	Small systems	Open source	N/A	N/A
ShaRK [157]	Control applications	Open source	Microkernel	Unix-like
TinyOS [158]	Wireless sensor network	Open source	N/A	N/A

As shown in the fourth column of Table 8.1, most investigated RTOSes can be classified into four categories in terms of kernel types. Microkernel provides only basic functionalities, with other services delivered by user-space servers. While monolithic kernel defines a high-level virtual interface on the top of hardware, with OS services implemented in supervisor mode modules by system calls. Hybrid kernel is a mixed kernel structure similar to a microkernel, but implemented as a monolithic kernel. Picokernel is very small microkernel, also called nanokernel.

Among these RTOSes, open source ones are preferred, because their source code are fully open to the public and there are plenty of supports from the thriving user communities. Four open source RTOSes are chosen for a further comparison, due to the fact that each of them

Table 8.2 Comparison in term of memory size and multiprocessing support

RTOS	Memory footprint	Multiprocessing support
eCos	tens to hundreds of KB	for some given processors
FreeRTOS	about 10KB for 8 bit processors	no
Micro C/OS-II	minimal: 3-10KB ROM; 2KB RAM	no
RTEMS	application dependent, <100KB	homogeneous and heterogeneous mulitprocessing

Table 8.3 Comparison in term of ported devices and specific features

RTOS	Porting complexity	Specific features
eCos	High	high configurability with a friendly configuration tool; many APIs support
FreeRTOS	Medium	small memory footprint; good demo applications
Micro C/OS-II	Low	small memory footprint; satisfy stringent real-time requirements
RTEMS	High	homogeneous and heterogeneous multiprocessing support; satisfy stringent real-time requirements

has reasonable porting complexity, has been ported to a large range of processors, and has good documents and the compatibility of GNU compilers which the RICA compiler is based on. Tables 8.2 and 8.3 give a detailed comparison for the four RTOSes. As shown in these tables, Micro C/OS-II has smallest memory requirements and lowest porting complexity, as its kernel clearly and concisely written in several thousand lines. Its simplicity also makes the interrupt and task switching latencies deterministic and debugging easier. Although Micro C/OS-II does not support multiprocessing, the basic knowledge about RTOS (e.g. scheduling and context switch) can be easily learned through porting this simple RTOS to RICA and programming multitasking based WiMAX on the top of it. Also as described in Chapter 4, the proposed multi-core architecture is master-slave based. Micro C/OS-II can be ported to this proposed multi-core system in a way where the functions of the kernel can only be executed on the master, with the slaves working as I/O processors. Therefore, Micro C/OS-II is chosen as the RTOS to be ported to the RICA architecture.

8.3 Porting Micro C/OS-II to RICA

From the system point of view, an embedded system can be simply described as a combination of the hardware infrastructure, an operating system and application programs. Figure 8.1 shows a diagram for a Micro C/OS-II based system. Acting as a middle layer between the hardware layer and the software layer, the RTOS Micro C/OS-II hides the details of the underlying hardware infrastructure from the software. For porting this RTOS to the RICA architecture, there are some requirements from both hardware and software sides [147].

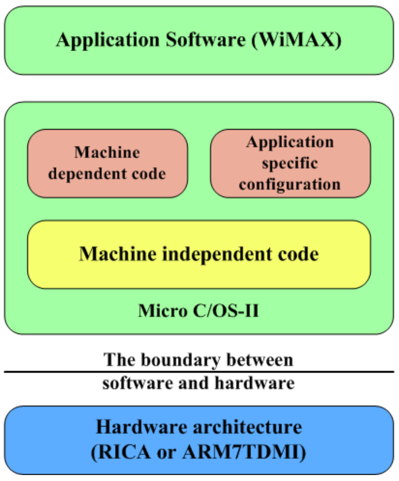

Application Software (WiMAX)
Machine dependent code
Application specific configuration
Machine independent code
Micro C/OS-II
The boundary between software and hardware
Hardware architecture (RICA or ARM7TDMI)

Figure 8.1 The Micro C/OS-II based system

8.3.1 Hardware requirements

The minimum hardware requirement for running Micro C/OS-II on the RICA architecture is shown in Figure 8.2. This hardware system is different from the standard RICA architecture shown in Figure 3.1. The data memory address space contains an interrupt vector table starting from address FFFFFF00, with a size of 80h. Upon the address of FFFFFF80, locates memory mapped registers for peripherals. This external register bank includes Timer0 control/status register, the interrupt control register, the interrupt status register and the previous program counter. The RTOS system time is provided by the system timer Timer0, which delivers an interrupt periodically to the RICA core as a tick. The tick period can be

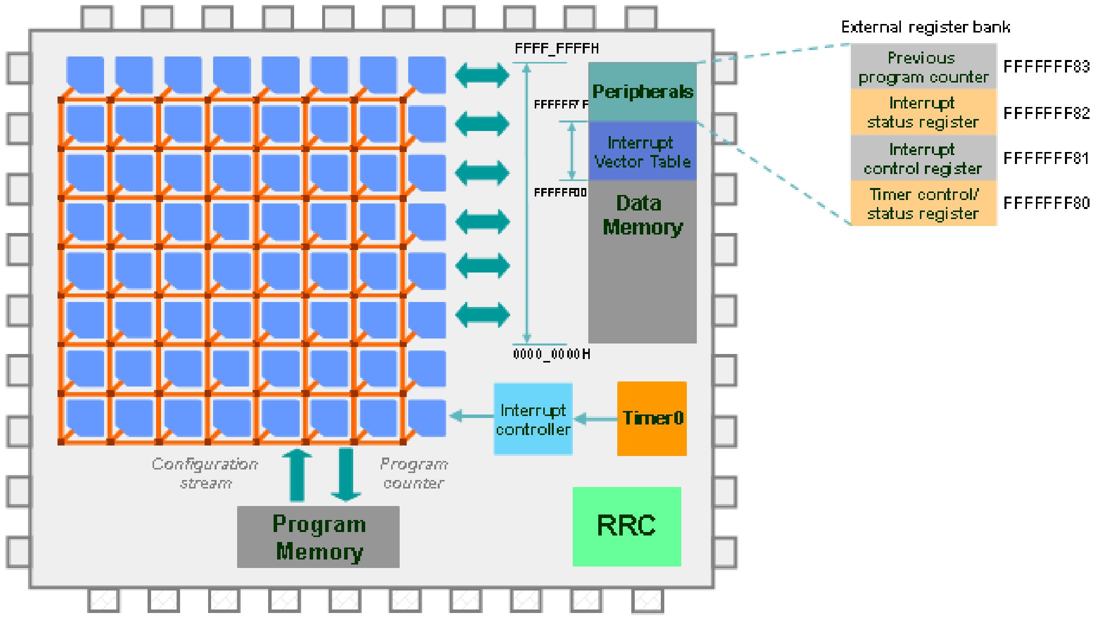


Figure 8.2 Hardware requirements for porting Micro C/OS-II to RICA

programmed by writing a value in the Timer0 control/status register. The interrupt controller is programmable through setting the interrupt control register, while its status can be read from the interrupt status register. The previous program counter is used in the interrupt handler for restoring the program counter of the interrupted task during task switching.

8.3.2 Software requirements

From the operating system point of view, several machine dependent files are required to describe the profile of the underlying hardware architecture. Described in Table 8.4, these

Table 8.4 Micro C/OS-II machine dependent files for RICA

Machine dependent files	Description
OS_CFG.H	System configuration
OS_CPU.H	Define the data types related to the processor
OS_CPU_C.C	Main functions: OSStartHighRdy(): enable the execution of the highest priority task OSCtxSw(): perform a context switch for the task pre-emption OSIntCtxSw(): perform a context switch for interrupts OSTickISR(): Timer0 tick interrupt service routine OSTaskStkInit(): initialise the stack frame of a task and other functions for functionality extension

files have been written according to the format defined by Micro C/OS-II. Some architecture-specific functions, such as initialisation, interrupt handling and task switching, are defined in these machine dependent files. Several Micro C/OS-II based programs have been written to test the basic RTOS functionalities like task management and scheduling. As for ARM7TDMI [159], the machine dependent files can be downloaded from the Micro C/OS-II official website.

8.4 The multitasking implementation of WiMAX on Micro C/OS-II

For demonstrating multitasking with Micro C/OS-II, a BPSK based WiMAX physical layer program has been modified to use APIs provided by this RTOS. For self-test' sake, the multitasking WiMAX program includes the transmitter, the receiver and an additional error checking function, as shown in Figure 8.3. The transmitter output data are fed as the input to the receiver, and the error checking function makes sure that the received data matches the sent data. Under the RTOS environment, the WiMAX program is partitioned into six Micro C/OS-II tasks: channel coding, OFDM downlink, OFDM uplink, demodulation, channel decoding and error checking, as shown in Figure 8.3. This partition is slightly different to that seen in Chapters 6 and 7, since the task balance is not the concern for multitasking on a single processor.

The multitasking scheme provided by this RTOS makes the tasks operate in a pipelined

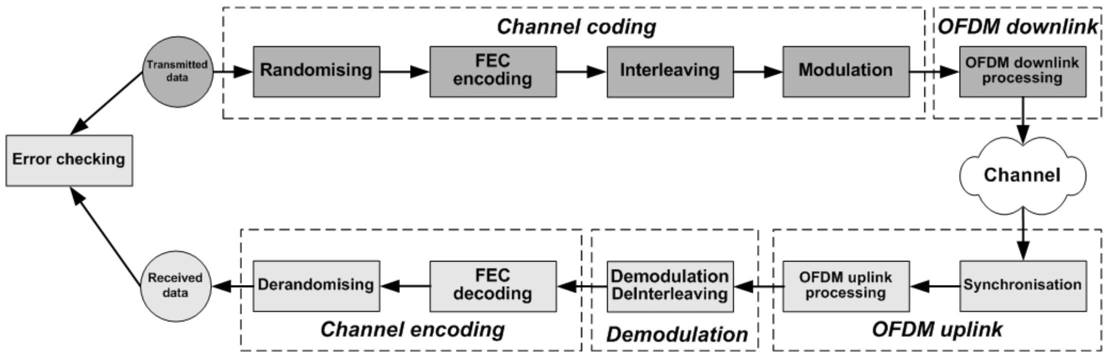


Figure 8.3 Multitasking WiMAX function blocks

fashion. It gives the tasks more independence from each other. These tasks have different priorities from high to low according to the above task order, since Micro C/OS-II's pre-emptive kernel does not support multiple tasks with a same priority [147]. In a pre-emptive multitasking OS, when a task with a higher priority is ready to run, the scheduler will interrupt the current running task and enable the pre-emptive task to perform. For achieving this, a context switch carries out to store the state of the interrupted task (e.g. the values of registers) into this task's stack and restore the state of the pre-emptive task from its stack. In Micro C/OS-II, a context switch may happen when an interrupt arrives as well. In this situation, the corresponding Interrupt Service Routine (ISR) saves the interrupted task's state. After the ISR finishes, a context switch function is called to restore the state of the ready Highest Priority Task (HPT) or the state of the interrupted task [147].

To protect the shared data between tasks and make tasks more independent from each other, the predefined RTOS semaphores are used as the interprocess synchronisation approach. Usually, the execution of the WiMAX program on Micro C/OS-II starts with the OS initialisation, followed by the creation of tasks and semaphores. Then the OS chooses the task with the highest priority to run. Tasks communicate with each other via semaphores. The program flow for the multitasking WiMAX on Micro C/OS-II is illustrated in Figure 8.4.

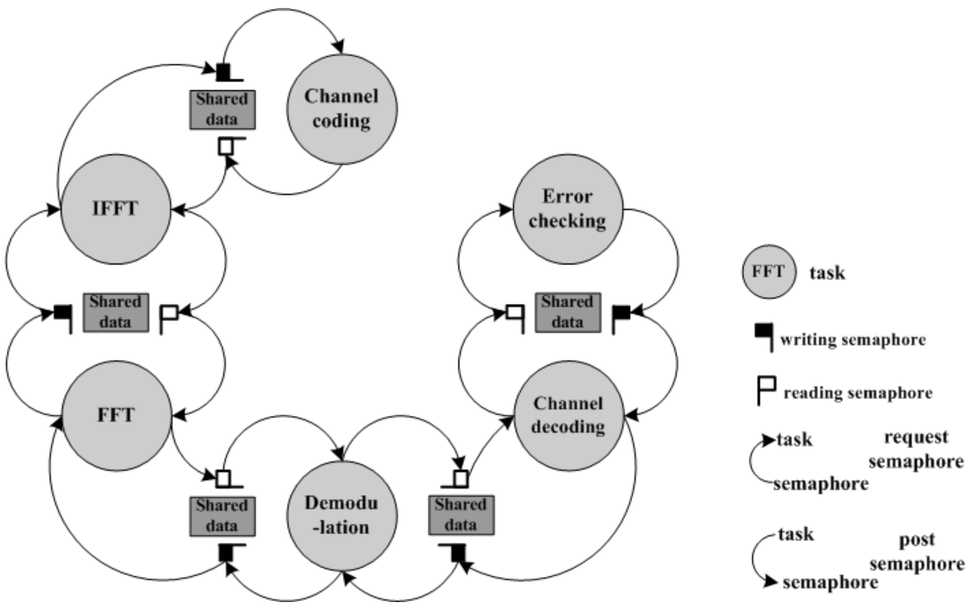


Figure 8.4 Program flow chart of WiMAX running on Micro C/OS-II

As shown in Figure 8.4, an arrow starting from a semaphore and ending at a task represents the task requesting the semaphore, while an arrow starting from a task and ending at a semaphore represents the task posting the semaphore. For each block of the shared data, there are two binary semaphores. One controls write access, while the other controls read access. Initialised as either 1 or 0, a binary OS semaphore ensures that only one task can access the shared resource at any one time [147]. The writing semaphores are initialised to 1, and the reading semaphores are initialised to 0. When the current task intends to access the shared data, either for writing or reading, it must first check the availability of the corresponding semaphore. If the value of that semaphore is zero, the current task will suspend, and the OS will place it back into the ready state until the semaphore becomes available. Each time when the current task suspends, the OS chooses to run the HPT from the ready list. Basically, for each OFDM symbol processing, a task can read the data shared with a previous task only after the previous task posts the reading semaphore.

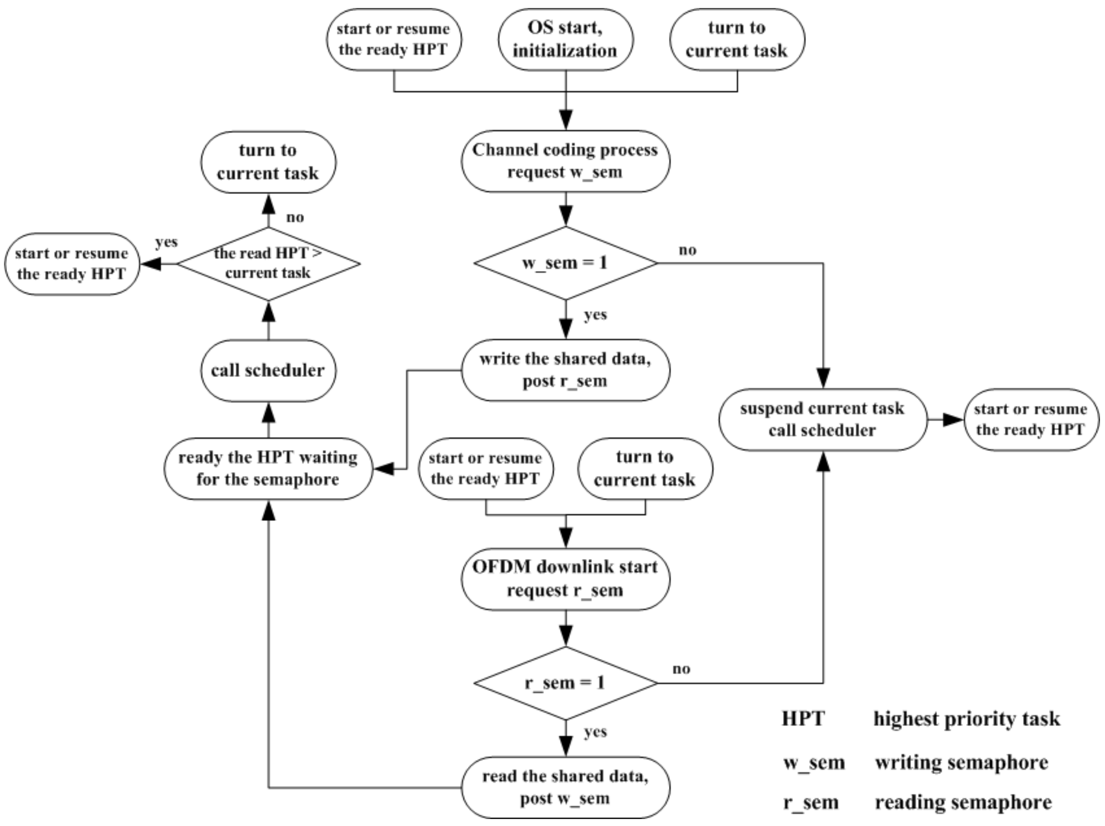

start or resume the ready HPT
OS start, initialization
turn to current task
Channel coding process request w_sem
turn to current task
no
yes
start or resume the ready HPT
the read HPT > current task
w_sem = 1
no
yes
write the shared data, post r_sem
call scheduler
suspend current task call scheduler
start or resume the ready HPT
ready the HPT waiting for the semaphore
start or resume the ready HPT
turn to current task
OFDM downlink start request r_sem
r_sem = 1
no
yes
read the shared data, post w_sem
HPT highest priority task
w_sem writing semaphore
r_sem reading semaphore

Figure 8.5 Detailed program flow diagram for the execution of channel coding and OFDM downlink tasks

Figure 8.5 illustrates the detailed program flow diagram for the execution of the channel coding task and the OFDM downlink task, giving an example of how the tasks run and switch. As shown in Figure 8.5, after OS initialisation and resource creation, the OS chooses the channel coding task to run for processing the first symbol, since it has the highest priority. Since the writing semaphore for the shared data between the channel coding task and the OFDM downlink task is initialised to 1, the channel coding task can write the shared data, and post the reading semaphore right after it finishes writing. During the semaphore post, the HPT waiting for this semaphore is made ready to run. The HPT is the OFDM downlink task in this case. Then the scheduler is called to check whether the ready HPT has a higher priority than the current running task. If so, a task based context switch happens and the current task will be pre-empted by the HPT. Otherwise, the semaphore posting function returns to the current running task. Obviously, at this time, the posting function returns to the channel coding task.

Next the channel coding task starts to process the second symbol after it finishes the computation of the first symbol. When the writing semaphore is reached for the second time, the channel coding task recognises that this semaphore is not available, since the OFDM DOWNLINK task has not start yet and read the shared data written there for the first symbol. Therefore, the channel coding task suspends, and will be returned to the ready state until the writing semaphore is released. After that, the OS scheduler chooses to run the OFDM downlink task, which is the highest priority ready task at that instant. Then the OFDM downlink task starts to run and requests the reading semaphore for the shared data between it and the channel coding task, which was posted during the execution of the channel coding task. After reading the shared data, the OFDM downlink task posts the writing semaphore, and the OS makes the channel coding task ready. Having higher priority, the channel coding task pre-empts the OFDM downlink task and continues to process the second symbol.

After processing the second symbol, the channel coding task again suspends due to the writing semaphore being unavailable. Then the scheduler makes the OFDM downlink task continue to finish its job for the first symbol and post the reading semaphore which makes the OFDM uplink task ready. Again the scheduler keeps the OFDM downlink task continuing to run. The OFDM downlink task is blocked at its processing for the second symbol as the channel coding task was before. At this time, the scheduler chooses the OFDM uplink task to run for the first symbol. Similar logic applies to the other tasks in the pipeline mode. Once all six tasks finish, the scheduler activates an idle task, which is a system task with the lowest priority.

8.5 Results

Firstly, to demonstrate the overheads introduced by porting Micro C/OS-II to the standard RICA, both the RICA implementations of a WiMAX program with and without this RTOS support have been developed. The detailed results are shown in Table 8.5, where WiMAX and RTOS represent the execution of WiMAX on a standard RICA with no OS support and with Micro C/OS II support, respectively. The RRC period of RICA was set to 2ns and the data memory access delay was set to 4ns. The comparison is based on the execution time per OFDM symbol, data memory and program memory requirements. The italic values represent the overheads introduced by the OS. Clearly, the overhead for the execution time is very small. The overhead for data memory is caused by the data structures for the OS resources and the stack for the tasks. Obviously, it is unavoidable that the program size significantly rises with the OS involved. As 1K configurable bits for each step, the increase is acceptable. Because both WiMAX and Micro C/OS II are not big programs, as some other programs (e.g. H.264) may cost thousands of steps executed on RICA.

Table 8.5 Results of WiMAX with/without RTOS support

Scheme	Execution time per symbol (μs)	Data memory (byte)	Program memory (step)
WiMAX	1,748	63,194	204
RTOS	1,779 ***1.8%***	90,051 ***42.5%***	396 ***94.1%***

For comparison's sake, the multitasking based WiMAX application has also been ported to ARM7TDMI which is popularly used for RTOS multitasking in embedded systems. ARM7TDMI is a 32-bit RISC embedded processor with a three-stage instruction pipeline [159]. For the ARM based implementation, this program was developed using the ARM Developer Suite [160] and was simulated on the Armulator (an instruction set simulator for ARM processors). The RRC period of the RICA processor was set as 2ns, while the ARM7TDMI run at a clock frequency of 500MHz. The comparison results are shown in Table 8.6, including the execution time on both simulators, as well as data memory and program memory requirements. In order to study the effect of memory latency on the execution time, different memory access delays were set. These delays are given as the values in the parentheses of the second column. In addition, with proper setting up, both of the simulations can quit when the idle task is reached.

As shown in Table 8.6, in terms of the execution time, the savings of the RICA architecture compared with ARM7TDMI are 53.6%, 70% and 78.8% for the memory latencies of 2ns, 4ns and 8ns, respectively. This is because the RICA architecture has more hardware resources available for use at any one time. With proper scheduling, it allows for the execution of multiple independent and dependent instructions concurrently in a single configuration context. Also, RICA performs much fewer memory accesses than ARM7TDMI. This is again a result of the higher availability of computation resources. Once an instruction is executed, instead of storing the result back to registers or to memory like a

Table 8.6 Comparison of multitasking WiMAX on RICA and ARM7TDMI

Architecture	Execution time per symbol (μs)	Data memory (byte)	Program memory (bit)	Area excluding memory (mm^2)
RICA	1,451 (2ns) 1,779 (4ns) 2,184 (8ns)	90,051	396 x 1 K	1.01
ARM7TDMI	3,127 (2ns) 5,945 (4ns) 10,306 (8ns)	86,338	69,280	0.59

normal processor does, the value can be directly fed into another instruction cell that performs a dependent instruction.

Basically, at the clock frequency of 500MHz, ARM7TDMI can complete a memory access in one clock cycle with 2ns memory latency, while having to insert one or three wait states with 4ns or 8ns memory latency, respectively. However, the specific features of RICA allow it to execute the memory access instructions concurrently with other instructions. No wait states need to be inserted to wait for the response from the memory, so long as there are sufficient other instructions that can be executed in the mean time. Therefore, the execution time on RICA depends much less on the memory latency, compared with ARM7TDMI which demonstrates an increase in execution time nearly proportional to any increase in memory latency. As shown in Table 8.6, the instruction cell array of a standard RICA has a nearly double area cost as an ARM7TDMI. It is mainly caused by switch boxes which are used as the interconnection and occupy about 50% area of the array.

The RICA architecture has a slightly larger data memory and about 5.7 times program memory requirement due to 1K bits being required for each step configuration. However, with the addition of suitable code compression techniques especially designed for the RICA architecture, this overhead can be dramatically reduced. Such techniques include the path-encoding technique proposed in [161] which can archive around 0.2 compression ratio and a dictionary-based lossless technique with a compression ratio in the range of 0.32 to 0.44 [162].

8.6 Summary

This chapter has covered the design of a multitasking WiMAX on the RICA architecture with an RTOS support. Through observing the overheads introduced by Micro C/OS-II, it has been demonstrated that the overhands were acceptable to enable an RTOS supported multitasking of a practical application, such as the WiMAX physical layer, on a single RICA processor. The comparison results showed that the standard RICA processor had much better performance than ARM7TDMI, especially when used with higher latency memory where up to a 78.8% saving in the execution time was achieved. Meanwhile this work laid a foundation for further exploration of multithreading on the proposed multi-core architecture.

Chapter 9
Conclusions

9.1 Introduction

This chapter concludes this thesis. In Section 9.2, the contents of individual chapters are reviewed. Section 9.3 lists some specific conclusions that can be drawn from the research in this thesis. Finally in Section 9.4, some possible directions for future work are addressed.

9.2 Review of thesis contents

Chapter 2 provided the background knowledge about broadband access technologies and WiMAX. Chapter 3 introduced multi-core processors and reconfigurable computing, described a newly emerging coarse-grained DR processor – RICA, and gave a review of the existing literature which is relevant to this thesis.

In Chapter 4, a novel master-slave based multi-core architecture was proposed, using RICA as the basic processing core. This architecture can support a wide range of memory architectural options, including shared/local data memory, shared register file and stream buffer. Meanwhile a multi-bank register file architecture was presented for alleviating the high competition for accessing the shared memory. A custom register file interface cell was designed to access the register file. Moreover this multi-core architecture contains many other components such as arbiters for memory blocks, interrupt controllers for processing cores, crossbar switches and a router. In addition, this architecture provides two synchronisation methods (i.e. spinlock and semaphore) for inter-processor communications by supporting a pair of atomic options LL/SC. In the end, Chapter 4 introduced a custom cell generation environment for integrating custom instruction cells into the RICA architecture.

In Chapter 5, a SystemC TLM based trace-driven simulator, called MRPSIM was presented to model this proposed multi-core architecture. This simulator provides a variety of architectural and performance analysis options, and can accurately simulate applications on multi-core architectures at a fast simulation speed. In addition, Chapter 5 introduced a preprocessing tool, Mpsockit, which can efficiently compress trace files and modify MRPSIM input files according to the requirement from different mapping methods. Several applications were chosen as test benches to estimate this simulator's performance. The results demonstrated that MRPSIM simulator could provide simulation speeds up to 300 KCPS, 300 KIPS and 25 KSPS for certain applications, respectively.

Chapter 6 focused on homogeneous multi-core solutions for WiMAX. A profiling-driven mapping methodology was proposed. This methodology incorporates task partitioning, task transformation and memory architecture aware data mapping. Three task partitioning methods, including task merging, task replication and loop level partitioning, were addressed and used for a BPSK based fixed WiMAX application. Furthermore, Chapter 6 introduced how to develop a C based multi-core project on the proposed multi-core architecture as well as how to synchronise tasks. At the end of Chapter 6, various homogeneous multi-core solutions, which combine different task partitioning methods and memory architectures, were developed for WiMAX. The impact of task partitioning methods and memory architectures on the system performance was analysed. The simulation results demonstrated up to 7.3 and 12 speedups or 3,241 Kbps and 659 Kbps throughputs could be achieved by employing eight and ten RICA cores for the WiMAX transmitter and receiver respectively.

In Chapter 7, a design space exploration methodology was proposed to efficiently search design spaces to find good solutions under certain system constraints. This algorithm is suitable for both single-core and multi-core designs. Instead of speedup and parallel efficiency, the ratio of throughput to area was used to estimate the efficiency of a multi-core design. In addition, Chapter 7 introduced various timing optimisation techniques to increase the throughput, and investigated area optimisation techniques to improve area efficiency without breaking the workload balance or worsening the overall throughout. Diverse designs were developed by means of the design space exploration methodology for both WiMAX transmitter and receiver. Totally eleven types of custom RICA cores were designed in these designs. The results demonstrated that a heterogeneous multi-core architecture involving totally nine RICA cores was the best solution. This solution can provide throughputs of up to 8.7 Mbps and 2.4 Mbps for WiMAX transmitter and receiver, respectively. In addition, Chapter 7 compared this best design with other multi-core based WiMAX designs. The comparison showed that the best design in Chapter 7 could deliver a very good throughput at relatively low area cost and performed even better than some commercial solution.

In Chapter 8, an RTOS Micro C/OS-II was ported to the RICA architecture. The multitasking implementation of WiMAX with this RTOS support was developed. The results demonstrated that the overheads introduced by Micro C/OS-II were acceptable. Furthermore, Chapter 8 compared the performance of the multitasking based WiMAX on both a standard RICA processor and an ARM7TDMI processor. The comparison results showed that a standard RICA processor could have up to a 78.8% saving in the execution time compared to ARM7TDMI.

9.3 Specific findings

This section presents a variety of conclusions, which stem from the research in this thesis.

Most academic and industry efforts on multi-core architectures have focused on using traditional processing cores like GPP core and DSP core to build multi-core architectures. This thesis investigated multi-core architectures building upon coarse-grained dynamically reconfigurable processing cores - an area that as yet has been little explored. The results in Chapters 6 and 7 showed that based on the proposed multi-core architecture, developed multi-core WiMAX solutions can deliver high performance beyond requirements of WiMAX standards. It proved that it was very promising to utilise DR cores in future high performance multi-core systems designed for applications like WiMAX.

In Chapter 5, a trace-driven simulator MRPSIM was proposed. By means of splitting the execution time into static and dynamic time as well as decoupling communication from computation, MRPSIM owns the benefits of both trace-driven and execution-driven simulators, in the mean time does not suffer the drawbacks of the two types of simulators. The advantages of MRPSIM include trace reuse, fast simulation and execution-driven like cycle accuracy. In addition, based on System TLM, MRPSIM simulator provides a platform for hardware/software co-design and enables fast application verification and architecture analysis.

In Chapter 6, a profiling-driven mapping methodology has been proposed to partition the target application into multiple tasks, schedule and map tasks on the proposed multi-core architecture. Three partitioning methods were investigated for WiMAX applications and suitable for different hardware environments. Task merging can be applied for a multi-core system with limited computing and storage resources, while task replication can bring into full play if the system has enough resources. For dedicated multi-core systems and applications, loop level partitioning can be used to explore the instruction level parallelism in order to improve the utilisation rates of processing cores. The results demonstrated that even based on unoptimised standard RICA cores, the best homogeneous multi-core solution presented in this chapter can deliver a high throughput satisfying the protocol requirement.

Chapter 7 presented a design space exploration methodology which can be used to find good solutions for both single-core and multi-core designs. This algorithm targets the optimisation of not only basic performance metrics like throughput and area, but also the ratio of throughput to area, through timing and area optimisation as well as the workload balance checking. By means of this exploration methodology, a variety of single-core and multi-core designs have been developed. The results proved that heterogeneous multi-core solutions with dynamically reconfigurable cores customised for dedicated tasks performed much better than homogeneous multi-core solutions, even though heterogeneous systems take more workloads. Meanwhile the comparison with other multi-core solutions demonstrated that the best homogeneous and heterogeneous multi-core solutions can provide a very good performance with a few simple DR processors, even based on a process technology one or two generations behind other solutions. It is mainly because of the salient characteristics of the RICA architecture and the low idle rate achieved by proper task partitioning and mapping.

Chapter 8 investigated an RTOS porting and multitasking WiMAX on a single RICA processor. It has been demonstrated that the overheads introduced by Micro C/OS-II were acceptable for running an RTOS on an RICA processor. Meanwhile, even with a limited instruction cells, a standard RICA processor delivered a much better performance for an RTOS based multitasking than an ARM7TDMI processor broadly used in embedded systems. Through this work in Chapter 8, valuable experience and knowledge were gained for further investigation of multithreading and RTOS support on the proposed multi-core architecture.

9.4 Directions for future work

The future work from this thesis can be split into three aspects: architecture, software tool flow and application.

From the architecture point of view, currently the proposed multi-core architecture has a simple interconnection and is difficult to scale to many-core architectures. The future work would improve the architecture by using more advanced interconnection techniques such as network-on-chip. In addition, this thesis chooses the RICA processor as the basic processing core. However, the proposed multi-core architecture is general enough to integrate other DR processing cores or even DSP cores. Therefore a heterogeneous multi-core architecture incorporating multiple processor families would be a research direction. In such architectures, processing cores from different families could make a close cooperation and meanwhile perform individual applications which they are particularly designed for. Another future work would be the hardware implementation of the multi-core architecture. In this thesis, the proposed multi-core architecture is modelled in TLM and performance results are gained from the high level simulation. Although this approach is suitable for fast architecture exploration and software verification, it is highly desired to implement this architecture in RTL to further prove the efficiency of the architecture and demonstrate its advantages. It could be done by either the re-design of the architecture in RTL or EDA tools which can transform designs from TLM to RTL. After that, an RTL simulation or even FPGA prototype could be carried out to demonstrate the hardware feasibility of the multi-core architecture.

The future work on the software tool flow would involve the improvement of MRPSIM simulator, more automated tools and dynamic scheduling supported by RTOSes. MRPSIM simulator could provide more architectural options (e.g. interconnection type and core type options) to keep up with the future development of the proposed multi-core architecture. Meanwhile the simulator could generate more detailed and user friendly performance analysis such as dynamic graphic display of memory accesses and interconnection traffic during simulation.

In Chapter 6, a mapping methodology has been proposed to partition, schedule and map tasks on multi-core architectures. In [163], we proposed an integer linear programming based approach to automatically map and schedule tasks on homogeneous multi-core architectures with RICA cores. A series of DSP applications have been implemented to demonstrate the efficacy of the technique. This approach could be improved to support RICA based heterogeneous multi-core architectures as well. A design space exploration methodology has been presented in Chapter 7. The operations (e.g. timing and area optimisation) in this algorithm could involve a lot of manual work. An automated tool would be needed to implement this algorithm. This design space exploration tool could be developed by using integer linear programming or genetic algorithm which can keep searching the design space to gradually approach the optimal solution.

Complementary to the static task scheduling performed by mapping tools, dynamical scheduling and RTOS support could enable the multi-core architecture to execute more general applications, besides deterministic applications like WiMAX. In Chapter 8, an RTOS Micro C/OS-II has been ported to RICA processor. Hence the next step on this direction would be porting this RTOS or another RTOS on the proposed multi-core architecture. Furthermore, how to support multithreading on the multi-core architecture could be investigated as well.

From the application side, this thesis mainly focuses on fixed WiMAX physical layer. However, as demonstrated in Chapter 5 and [163], many other applications have been executed on the proposed multi-core architecture as well. More applications are planned to be mapped to this multi-core architecture, including, but not limited to, WLAN physical layer, H.264 video compression, freeman demosaicing algorithm and the receiver in global positioning system. As for fixed WiMAX itself, there would be a lot of work for further optimisation and the implementation of more fixed WiMAX profiles such as QPSK and

64QAM based. In addition, as the mobile version of WiMAX, IEEE 802.16-2005 standard [1] is highly valued, especially after it became one of the four 3G standards in Oct. 2007. The upgrade from fixed WiMAX to mobile WiMAX would be investigated. For achieving it, more components would be required to be designed, such as Multi-Input Multi-Output (MIMO) and SOFDMA.

Reference

[1] "Air Interface for Fixed and Mobile Broadband Wireless Access Systems," IEEE Std. 802.16e-2005, 2005.

[2] "Air Interface for Fixed Broadband Wireless Access Systems," IEEE Std. 802.16-2004 (Revision of IEEE Std. 802.16-2001), 2004.

[3] G.S.V. Radha K. Rao and G. Radhamani, *WiMAX: A Wireless Technology Revolution*, 1 ed, Auerbach Publications, 2007.

[4] Wm. A. Wulf and Sally. A. McKee, "Hitting the memory wall: implications of the obvious," *SIGARCH Comput. Archit. News*, vol. 23, pp. 20-24, 1995.

[5] David W. Wall, "Limits of instruction-level parallelism," the Fourth International Conference on Architectural Support for Programming Languages and Operating systems, pp. 176-188, 1991.

[6] Peter Claydon, "Multicore future is right now," *EE Times-Asia*, 2007.

[7] Sami Khawam, Ioannis Nousias, Mark Milward, Ying Yi, Mark Muir, and Tughrul Arslan, "The Reconfigurable Instruction Cell Array," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, pp. 75-85, 2008.

[8] Fiona Vanier, "World Broadband Statistics: Q2 2008," Point Topic, September 2008.

[9] "Asymmetric Digital Subscriber Line (ADSL) transceivers - Extended bandwidth ADSL2 (ADSL2+)," ITU-T Recommendation G.992.5 Annex M Std., 2005.

[10] "DOCSIS 2.0 Specifications," Cable modem standard, CableLabs, 2002.

[11] "Broadband SoHo FTTx Tutorial," BroadbandSoHo, 2007.

[12] "Broadband over Power Line Networks: Medium Access Control and Physical Layer Specifications," IEEE P1901 Draft Standard.

[13] "Technical Specifications and Technical Reports for a UTRAN-based 3GPP system Release 5 and Release 6," 3rd Generation Partnership Project (3GPP) Technical Specification, 2007.

[14] "CDMA2000 High Rate Packet Data Air Interface Specification," 3GPP2 Std. TIA-856-B, 2006, http://www.3gpp2.org/.

[15] "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," IEEE Std. 802.11-2007 (Revision of IEEE Std. 802.11-1999), 2007.

[16] "EDGE, HSPA and LTE Broadband Innovation," Rysavy Research and 3G Americas, September 2008.

[17] "Satellite Internet access," Wikipedia, http://en.wikipedia.org/wiki/Satellite_Internet_access, 2008.

[18] Jeffrey G. Andrews, Arunabha Ghosh, and Rias Muhamed, *Fundamentals of WiMAX: Understanding Broadband Wireless Networking*, Prentice Hall, 2007.

[19] Ian A. Glover and Peter M. Grant, *Digital communications* 2ed, Prentice Hall, 2004.

[20] I. Reed and G. Solomon, "Polynomial codes over certain Finite Fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, pp. 300-304, June 1960.

[21] John Proakis, *Digital Communications*, 4th ed, McGraw-Hill, 2001.

[22] Matthew Gast, *802.11 Wireless Networks The Definitive Guide*, O'Reilly, April 2005.

[23] L. Rabiner and B. Gold, *Theory and application of Digital Signal Processing*, Prentice Hall, 1975.

[24] B. Guoan and E. Jones, "A pipelined FFT processor for word sequential data," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 37, pp. 1982–1985, December 1989.

[25] Martin Vetterli and Pierre Duhamel, "Split-radix algorithms for length-p^m DFT's," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, pp. 57–64, January 1989.

[26] "Implementing WiMAX PHY Processing Using the Freescale MSC8126 Multi-Core DSP," Freescale Semiconductor, 2004.

[27] Xin Zhao, Ahmet T. Erdogan, and Tughrul Arslan, "OFDM Symbol Timing Synchronization System on a Reconfigurable Instruction Cell Array," Exploiting Loop-Level Parallelism on Multi-Core Architectures for the WiMAX Physical Layer, pp. 319-322, 2008.

[28] Jan-Jaap van de Beek, Magnus Sandell, and Per Ola Borjesson, "ML Estimation of Time and Frequency Offset in OFDM Systems," *IEEE Transactions on signal processing*, vol. 45, pp. 1800-1805, 1997.

[29] Andrew J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, pp. 260-269, 1967.

[30] Ahmed O. El-Rayis, Xin Zhao, Tughrul Arslan, and Ahmet T. Erdogan, "Dynamically programmable Reed Solomon processor with embedded Galois Field multiplier," 2008 International Conference on Field-Programmable Technology (FPT '08), pp. 269-272, 2008.

[31] J. Heikkinen, J. Sertamo, T. Rautiainen, and J. Takala, "Design of transport triggered architecture processor for discrete cosine transform," 15th Annual IEEE International ASIC/SOC Conference, pp. 87-91, 2002.

[32] H. Zhang, V. Prabhu, V. George, M. Wan, M. Benes, A. Abnous, and J. M. Rabaey, "A 1-V heterogeneous reconfigurable DSP IC for wireless baseband digital signal processing," *IEEE Journal of Solid-State Circuits*, vol. 35, pp. 1697-1704, 2000.

[33] T. Sato, H. Watanabe, and K. Shiba, "Implementation of dynamically reconfigurable processor DAPDNA-2," 2005 IEEE VLSI-TSA International Symposium on VLSI Design, Automation and Test (VLSI-TSA-DAT), pp. 323-324, 2005.

[34] Gregory V. Wilson, "The History of the Development of Parallel Computing," 1994.

[35] "Quad-Core Intel® Xeon® Processor 5400 Series 45nm Reference Guide," Intel Corporation, 2008.

[36] "NVIDIA GEFORCE GTX 200 GPU Datasheet," NVIDIA.

[37] Kazuyuki Hirata and John Goodacre, "ARM MPCore; The streamlined and scalable ARM11 processor core," 12nd Asia and South Pacific Design Automation Conference (ASP-DAC '07), pp. 747-748, 2007.

[38] M. Butts, A. M. Jones, and P. Wasson, "A Structural Object Programming Model, Architecture, Chip and Tools for Reconfigurable Computing," 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), pp.

55-64, 2007.

[39] Geoff Koch, "Discovering Multi-Core: Extending the Benefits of Moore's Law," White paper, Intel, 2005.

[40] A. P. Chandrakasan and R. W. Brodersen, *Low power digital CMOS design*, Kluwer, 1995.

[41] John L. Hennessy and David A. Patterson, *Computer Architecture: A Quantitative Approach* 3rd ed, Morgan Kaufmann, 2002.

[42] "Family 10h AMD Opteron™ Processor Product Data Sheet," Revision 3.02, Advanced Micro Devices, June 2009.

[43] "IBM dual-core Power6," IBM, 2007.

[44] William Bryg and Jerome Alabado, "The UltraSPARC T1 Processor - High Bandwidth For Throughput Computing," White paper, Sun Microsystems, Inc., December 2005.

[45] "IBM PowerPC 970MP RISC Microprocessor Datasheet," Version 1.2, IBM, July 2007.

[46] "Intel® Core™2 Quad-Core Processor QX6000 and Q6000 Series Datasheet," Intel Corporation, Auguest 2007.

[47] "Family 10h AMD Phenom™ Processor Product Data Sheet," Revison 3.00, Advanced Micro Devices, November 2007.

[48] D. Pham, S. Asano, M. Bolliger, M. N. Day, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Riley, D. Shippy, D. Stasiak, M. Suzuoki, M. Wang, J. Warnock, S. Weitzel, D. Wendel, T. Yamazaki, and K. Yazawa, "The design and implementation of a first-generation CELL processor," Digest of Technical Papers, 2005 IEEE International Solid-State Circuits Conference (ISSCC '05) pp. 184-592 Vol. 1, 2005.

[49] Gajinder Panesar, Daniel Towner, Andrew Duller, Alan Gray, and Will Robbins, "Deterministic parallel processing," *International Journal of Parallel Programming* vol. 34, pp. 323-341, 2006.

[50] Andrew Duller, Gajinder Panesar, and Daniel Towner, "Parallel Processing — the picoChip way!," *Communicating Process Architectures 2003*, pp. 125–138, 2003.

[51] B. Scott Michel, "GPGPU Computing and the Heterogeneous Multi-Core Future," HPCwire, December 01, 2006.

[52] Rick Merritt, "CPU designers debate multi-core future," EE times, June 2, 2008.

[53] Rakesh Kumar, M. Tullsen Dean, Ranganathan Parthasarathy, P. Jouppi Norman, and I. Farkas Keith, "Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance," *ACM SIGARCH Computer Architecture News* vol. 32, pp. 64, 2004.

[54] A. Beric, R. Sethuraman, C. A. Pinto, H. Peters, G. Veldman, P. van de Haar, and M. Duranton, "Heterogeneous multiprocessor for high definition video," 2006 International Conference on Consumer Electronics (ICCE '06), pp. 401-402, 2006.

[55] Shee Seng Lin and S. Parameswaran, "Design Methodology for Pipelined Heterogeneous Multiprocessor System," 44th ACM/IEEE Design Automation Conference (DAC '07) pp. 811-816, 2007.

[56] Hiroaki Shikano, Yuki Suzuki, Yasutaka Wada, Jun Shirako, Keiji Kimura, and Hironori Kasahara, "Performance Evaluation of Heterogeneous Chip Multi-Processor with MP3 Audio Encoder," IEEE Symposium on Low-Power and High Speed Chips (COOL Chips IX), pp. 349-363, Apr. 2006.

[57] Xtensa Processor, Tensilica Inc. (htttp://www.tensilica.com).

[58] "Intel® Core™2 Duo Processor E8000 and E7000 Series Datasheet," Intel Corporation, June 2009.

[59] "Intel® IXP2350 Network Processor Product Brief," Intel Corporation, 2005.

[60] MRC6011, Freescale, 2004, http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MRC6011&nodeId=012795LCWs.

[61] D. Iancu, H. Ye, E. Surducan, M. Senthilvelan, J. Glossner, V. Surducan, V. Kotlyar, A. Iancu, G. Nacer, and J. Takala, "Software Implementation of WiMAX on the Sandbridge SandBlaster Platform," *LECTURE NOTES IN COMPUTER SCIENCE*, vol. 4017, pp. 435-446 2006.

[62] Compton Katherine and Hauck Scott, "Reconfigurable computing: a survey of systems and software," *ACM Computing Surveys*, vol. 34, pp. 171-210, 2002.

[63] Sami Khawam, "Domain-specific and reconfigurable instruction cells based architectures for low-power SoC " Ph.D. thesis, University of Edinburgh, 2006.

[64] Paul M. Heysters, Gerard J.M. Smit, and Egbert Molenkamp, "Montium - Balancing between Energy-Efficiency, Flexibility and Performance," International Conference on Engineering of Reconfigurable Systems and Algorithms, pp. 235-241, 2003.

[65] Hartej Singh, Ming Hau Lee, Guangming Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. Chaves Filho, "MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE Transactions on Computers*, vol. 49, pp. 465-481, 2000.

[66] Nikolaos S. Voros and Konstantinos Masselos, *System Level Design of Reconfigurable Systems-on-Chip*, 1 ed, Springer, 2005.

[67] R. Tessier and J. Rose I. Kuon, "FPGA Architecture: Survey and Challenges," *Foundations and Trends in Electronic Design Automation*, vol. 2, pp. 135-253, 2008.

[68] "Virtex-5 User Guide 4.5," Xilinx, San Jose, 2009.

[69] "Stratix-III Device Handbook," Altera, San Jose, 2008.

[70] Hartenstein Reiner, "Coarse grain reconfigurable architecture (embedded tutorial)," 6th Asia and South Pacific Design Automation Conference (ASP-DAC '01), 2001.

[71] "Adapt2400 ACM Architecture Overview," QuickSilver Technologies, 2004.

[72] B. Mei, S. Vernalde, D. Verkest, H. D. Man, and R. Lauwereins, "ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix," Field-Programmable Logic and Applications, pp. 61-70, 2003.

[73] "HiveFlex CSP2000 Series Programmable OFDM Communication Signal Processor," databrief, Silicon Hive, Eindhoven, 2007.

[74] E. Mirsky and A. DeHon, "MATRIX: a reconfigurable computing architecture with

configurable instruction distribution and deployable resources," IEEE Symposium on FPGAs for Custom Computing Machines, pp. 157-166, 1996.

[75] "XPP64-A1 Reconfigurable Processor," Preliminary Datasheet, PACT XPP Technologies, Munich, 2003.

[76] A. K. W. Yeung and J. M. Rabaey, "A reconfigurable data-driven multiprocessor architecture for rapid prototyping of high throughput DSP algorithms," the Twenty-Sixth Hawaii International Conference on System Sciences, pp. 169-178 vol.1, 1993.

[77] H. Schmit, D. Whelihan, A. Tsai, M. Moe, B. Levine, and R. Reed Taylor, "PipeRench: A virtualized programmable datapath in 0.18 micron technology," IEEE 2002 Custom Integrated Circuits Conference, pp. 63-66, 2002.

[78] Ebeling Carl, C. Cronquist Darren, and Franklin Paul, "RaPiD - Reconfigurable Pipelined Datapath," the 6th International Workshop on Field-Programmable Logic, Smart Applications, New Paradigms and Compilers, 1996.

[79] J. R. Hauser and J. Wawrzynek, "Garp: a MIPS processor with a reconfigurable coprocessor," the 5th IEEE Symposium on FPGA-Based Custom Computing Machines, pp. 12-21, 1997.

[80] Marshall Alan, Stansfield Tony, Kostarnov Igor, Vuillemin Jean, and Hutchings Brad, "A reconfigurable arithmetic array for multimedia applications," the 1999 ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays, pp. 135 - 143 1999.

[81] E. Waingold, M. Taylor, V. Sarkar, V. Lee, W. Lee, J. Kim, M. Frank, P. Finch, S. Devabhaktumi, R. Barua, J. Babb, S. Amarsinghe, and A. Agarwal, "Baring it all to Software: The Raw Machine," *IEEE Computer*, vol. 30, pp. 86-93, 1997.

[82] Miyamori Takashi and Olukotun Kunle, "REMARC: reconfigurable multimedia array coprocessor," the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays, 1998.

[83] "Virtex-6 Family Overview," Xilinx, San Jose, 2009.

[84] "Stratix II Device Handbook," Stratix, San Jose, 2007.

[85] Ying Yi, Ioannis Nousias, Mark Milward, Sami Khawam, Tughrul Arslan, and Iain Lindsay, "System-level scheduling on instruction cell based reconfigurable systems," The Conference on Design, Automation and Test in Europe (DATE '06), pp. 381 - 386, 2006.

[86] "Deliver WiMAX Faster - Enabling fast design-in, small form factors, and energy-efficient performance with an integrated WiMAX chipset solution," White paper, Intel, 2008.

[87] "WiMAX 3.5 GHz Mini-PCI Reference Design - EvolutiveTM WiMAX NP7256 Series," Wavesat, Montreal, 2007.

[88] "Intel® PRO/Wireless 5116 Broadband Interface," Product brief, Intel, 2005.

[89] Deepak Boppana, "FPGA-Based WiMAX System Design," Altera Corporation, 2005.

[90] H. Lai and S. Boumaiza, "WiMAX baseband processor implementation and validation on a FPGA/DSP platform," 2008 Canadian Conference on Electrical and

Computer Engineering, pp. 001449-001452, 2008.

[91] "Implementation of an OFDM Wireless Transceiver using IP Cores on an FPGA," white paper, Lattice Semiconductor Corporation, 2005.

[92] A. Sghaier, S. Areibi, and R. Dony, "IEEE802.16-2004 OFDM functions implementation on FPGAS with design exploration," International Conference on Field Programmable Logic and Applications, pp. 519-522, 2008.

[93] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, pp. 203–215, 2007.

[94] "PC7218 Baseband Reference Design Product Brief v0.3," picoChip, 2006.

[95] "Software Reference Design PC8520 802.16-2004 OFDM256, Basestation PHY," picoChip, 2006.

[96] "MSC8126 data sheet revision 14," Freescale, 2008.

[97] Qing Wang, Da Fan, Jianwen Chen, Yonghua Lin, Zhenbo Zhu, and Xiaoyan Dang, "WiMAX BS Transceiver Based on Cell Broadband Engine," 4th IEEE International Conference on Circuits and Systems for Communications (ICCSC 2008) pp. 182-186, 2008.

[98] "Intel NetStructure WiMAX Baseband Card Product Brief," Intel Corporation, 2006.

[99] John Glossner, Mayan Moudgill, Daniel Iancu, Gary Nacer, Sanjay Jintukar, Stuart Stanley, Michael Samori, Tanuj Raja, and Michael Schulte, "The Sandbridge Sandblaster Convergence Platform," White paper, Sandbridge Technologies, White Plains, 2005.

[100] Geoffrey F. Burns, Marco Jacobs, Menno Lindwer, and Bertrand Vandewiele, "Silicon Hive's Scalable and Modular Architecture Template for High-Performance Multi-Core Systems," Silicon Hive.

[101] Irv Englander, *The Architecture of Computer Hardware and Systems Software: An Information Technology Approach* 3rd ed, John Wiley and Sons, 2003.

[102] V. Zyuban and P. Kogge, "The energy complexity of register files," 1998 International Symposium on Low Power Electronics and Design, pp. 305-310, 1998.

[103] V. Zyuban and P. M. Kogge, "Inherently lower-power high-performance superscalar architectures," *IEEE Transactions on Computers*, vol. 50, pp. 268-285, 2001.

[104] Jos Lorenzo Cruz, Antonio Gonzalez, Mateo Valero, and Nigel P. Topham, "Multiple-banked register file architectures," 27th Annual International Symposium on Computer Architecture (ISCA '00), pp. 316 - 325, 2000.

[105] I. Park, M. D. Powell, and T. N. Vijaykumar, "Reducing register ports for higher speed and lower energy," 35th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-35), pp. 171-182, 2002.

[106] B.K. Khailany, T. Williams, J. Lin, E.P. Long, M. Rygh, D.W. Tovey, and W.J. Dally, "A Programmable 512 GOPS Stream Processor for Signal, Image, and Video Processing," *IEEE Journal of Solid-State Circuits*, vol. 43, pp. 202-213, Jan. 2008.

[107] Andrew S. Tanenbaum and Albert S. Woodhull, *Operating systems : design and implementation*, 3rd ed, Pearson Prentice Hall, 2006.

[108] David E. Culler, Jaswinder Pal Singh, and Anoop Gupta, *Parallel computer*

architecture: a hardware/software approach, Morgan Kaufmann, 1999.

[109] A. Rose, S. Swan, J. Pierce, and J. Fernandez, "Transaction level modeling in SystemC," Open SystemC Initiative, 2005.

[110] Naraig Manjikian, "Multiprocessor enhancements of the SimpleScalar tool set," *ACM SIGARCH Computer Architecture News* vol. 29, pp. 8-15, 2001.

[111] C. J. Hughes, V. S. Pai, P. Ranganathan, and S. V. Adve, "Rsim: simulating shared-memory multiprocessors with ILP processors," *Computer*, vol. 35, pp. 40-49, 2002.

[112] Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *ACM SIGARCH Computer Architecture News* vol. 33, pp. 92-99, 2005.

[113] John D. Davis, Cong Fu, and James Laudon, "The RASE (Rapid, Accurate Simulation Environment) for chip multiprocessors," *ACM SIGARCH Computer Architecture News*, vol. 33, pp. 14-23, 2005.

[114] P. M. Ortego and P. Sack, "SESC: SuperESCalar Simulator," http://iacoma.cs.uiuc.edu/~paulsack/sescdoc/, 2004.

[115] Luca Benini, Davide Bertozzi, Alessandro Bogliolo, Francesco Menichelli, and Mauro Olivieri, "MPARM: Exploring the Multi-Processor SoC Design Space with SystemC," *Journal of VLSI Signal Processing Systems* vol. 41, pp. 169-182, 2005.

[116] Doug Burger and Todd M. Austin, "The SimpleScalar tool set, version 2.0," *ACM SIGARCH Computer Architecture News* vol. 25, pp. 13-25, 1997.

[117] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, pp. 50-58, 2002.

[118] M. Dales, "SWARM," http://www.dcs.gla.ac.uk/~michael/phd/swarm.html.

[119] Frank Ghenassia, *Transaction-level modeling with SystemC : TLM concepts and applications for embedded systems*, Springer, 2005.

[120] Jayaram Bhasker, *A SystemC primer*, 2nd ed, Star Galaxy Pub., 2004.

[121] Fei Sun, N. K. Jha, S. Ravi, and A. Raghunathan, "Synthesis of application-specific heterogeneous multiprocessor architectures using extensible processors," 18th International Conference on VLSI Design, pp. 551-556, 2005.

[122] Karkowski Ireneusz and Corporaal Henk, "Design space exploration algorithm for heterogeneous multi-processor embedded system design," 35th IEEE/ACM Design Automation Conference, pp. 82 - 87, 1998.

[123] E. Rockoff Todd, "An Analysis of Instruction-Cached SIMD Computer Architecture," Technical Report: CS-93-218, Carnegie Mellon University.

[124] "Using MicroC/OS II RTOS with the Nios II Processor Tutorial," Altera, 2007.

[125] Chang Ning Sun, "Nucleus for Xilinx FPGAs - A New Platform for Embedded System Design," Mentor Graphics, Wilsonville, 2004.

[126] Anthony J. Massa, *Embedded Software Development with eCos*, Prentice Hall, 2002.

[127] Michael Hohmuth and Hermann H¨artig, "Pragmatic nonblocking synchronization for real-time systems," the 2001 USENIX Annual Technical Conference

(USENIX'01), pp. 217-230, 2001.

[128] "RTEMS 4.9.1, Real-Time Executive for Multiprocessor Systems," http://www.rtems.com/, 2008.

[129] "NetBSD 4.0.1," http://www.netbsd.org/, 2008.

[130] Pawel Pisarczyk, "Experimental dependability evaluation of memory manager in the real-time operating system," *International journal of Computer Science and Applications*, vol. 4, pp. 33-38, 2007.

[131] "QNX Neutrino RTOS 6.4.0," QNX, Ottawa, 2008.

[132] "FreeRTOS v5.1.1," www.freertos.org/, 2008.

[133] "ThreadX 5.0," Express Logic, San Diego, 2006.

[134] "TRON 4.03.02 The Real-time Operating System Nucleus " http://www.tron.org/index-e.html, 2008.

[135] David McCullough, "Getting started with uClinux," www.uclinux.org.

[136] "Vxworks 6.7," Wind River, Alameda, 2009.

[137] "Windows XP Embedded," Microsoft, Redmond, 2001.

[138] "MontaVista Linux," MontaVista Software, Santa Clara, 2008.

[139] "Nucleus OS " Mentor Graphics, Wilsonville, 2005.

[140] "Palm OS 5.4.9," Palm, Sunnyvale 2006.

[141] "Prex Technology Overview version 1.2," http://prex.sourceforge.net, 2008.

[142] "Symbian OS 9.5," Symbian, Southwark 2008.

[143] "Windows Mobile 6.5," Microsoft, Redmond, 2009.

[144] "Windows CE 6.0 R2," Microsoft, Redmond, 2007.

[145] "Integrity 5.0," Green Hills, Santa Barbara, 2005.

[146] "LynxOS 4.2," LynxOS, San Jose, 2006.

[147] Jean J. Labrosse, *MicroC OS II: The Real Time Kernel* 2nd ed, CMP Books, 2002.

[148] "RTLinux 2.6," Wind River, Alameda, 2007.

[149] "Agnix 0.0.4," http://agnix.sourceforge.net/, 2005.

[150] "ChorusOS 5.0," www.experimentalstuff.com, 1997.

[151] Dunkels Adam, Gronvall Bjorn, and Voigt Thiemo, "Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors," the 29th Annual IEEE International Conference on Local Computer Networks, pp. 455 - 462, 2004.

[152] "ETLinux 1.1.3," http://www.etlinux.org/, 2006.

[153] "Freesco 0.4.0," www.freesco.org, 2008.

[154] "Inferno 4," Bell Labs, Murray Hill, 2007.

[155] "Nut/OS 4.0.3," http://www.ethernut.de, 2006.

[156] "pico]OS 1.0.1," http://picoos.sourceforge.net/, 2001.

[157] "SHaRK 1.5.4 Soft Hard Real-Time Kernel," http://shark.sssup.it/, 2007.

[158] "TinyOS 2.1," http://www.tinyos.net/, 2008.

[159] "ARM7TDMI Technical Reference Manual," ARM Limited, Cambridge, 2004.

[160] "ARM Developer Suite Developer Guide Version 1.2," ARM Limited, Cambridge, 2001.

[161] Tughrul Arslan, Ioannis Nousias, Sami Khawam, Mark Milward, M. Muir, and Y. Yi, "Encoding and Decoding Methods," Pending Patent, PCT/GB2008/000367, 2008.

[162] N. Aslam, M. J. Milward, A. T. Erdogan, and T. Arslan, "Code Compression and Decompression for Coarse-Grain Reconfigurable Architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, pp. 1596-1608, 2008.

[163] Ying Yi, Wei Han, Xin Zhao, Ahmet T. Erdogan, and Tughrul Arslan, "An ILP Formulation for Task Mapping and Scheduling on Multi-core Architectures," The Conference on Design, Automation and Test in Europe (DATE '09), pp. 33-38, 2009.