

Nadia Nedjah
Luiza de Macedo Mourelle

Co-design for System Acceleration

A Quantitative Approach

 Springer

Co-design for System Acceleration

A Quantitative Approach

CO-DESIGN FOR SYSTEM ACCELERATION

A Quantitative Approach

NADIA NEDJAH

Department of Electronics Engineering and Telecommunications,
State University of Rio de Janeiro, Brazil

LUIZA DE MACEDO MOURELLE

Department of Systems Engineering and Computation,
State University of Rio de Janeiro, Brazil



Springer

A C.I.P. Catalogue record for this book is available from the Library of Congress.

ISBN-13 978-1-4020-5545-4 (HB)

ISBN-13 978-1-4020-5546-1 (e-book)

Published by Springer,
P.O. Box 17, 3300 AA Dordrecht, The Netherlands.

www.springer.com

Printed on acid-free paper

All Rights Reserved

© 2007 Springer

No part of this work may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, microfilming, recording or otherwise, without written permission from the Publisher, with the exception of any material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work.

*To my mother and sisters,
Nadia*

*To my father (in memory)
and mother, Luiza*

Contents

| | |
|------------------------------------|------|
| Dedication | v |
| List of Figures | xi |
| List of Tables | xv |
| Preface | xvii |
| Acknowledgments | xix |
| 1. INTRODUCTION | 1 |
| 1.1 Synthesis | 2 |
| 1.2 Design Approaches | 3 |
| 1.3 Co-Design | 4 |
| 1.3.1 Methodology | 5 |
| 1.3.2 Simulation | 6 |
| 1.3.3 Architecture | 6 |
| 1.3.4 Communication | 6 |
| 1.4 Structure and Objective | 7 |
| 2. THE CO-DESIGN METHODOLOGY | 9 |
| 2.1 The Co-Design Approach | 10 |
| 2.2 System Specification | 11 |
| 2.3 Hardware/Software Partitioning | 12 |
| 2.4 Hardware Synthesis | 15 |
| 2.4.1 High-Level Synthesis | 16 |
| 2.4.2 Implementation Technologies | 17 |
| 2.4.3 Synthesis Systems | 20 |
| 2.5 Software Compilation | 21 |
| 2.6 Interface Synthesis | 22 |

| | | |
|-------|------------------------------------|----|
| 2.7 | System Integration | 23 |
| 2.8 | Summary | 27 |
| 3. | THE CO-DESIGN SYSTEM | 29 |
| 3.1 | Development Route | 30 |
| 3.1.1 | Hardware/Software Profiling | 31 |
| 3.1.2 | Hardware/Software Partitioning | 33 |
| 3.1.3 | Hardware Synthesis | 33 |
| 3.1.4 | Software Compilation | 36 |
| 3.1.5 | Run-Time System | 36 |
| 3.2 | Target Architecture | 37 |
| 3.2.1 | Microcontroller | 38 |
| 3.2.2 | Global Memory | 40 |
| 3.2.3 | Controllers | 41 |
| 3.2.4 | Bus Interface | 42 |
| 3.2.5 | The Coprocessor | 45 |
| 3.2.6 | The Timer | 45 |
| 3.3 | Performance Results | 45 |
| 3.3.1 | First Benchmark: PLUM Program | 46 |
| 3.3.2 | Second Benchmark: EGCHECK Program | 47 |
| 3.3.3 | Results Analysis | 48 |
| 3.4 | Summary | 50 |
| 4. | VHDL MODEL OF THE CO-DESIGN SYSTEM | 53 |
| 4.1 | Modelling with VHDL | 54 |
| 4.1.1 | Design Units and Libraries | 55 |
| 4.1.2 | Entities and Architectures | 55 |
| 4.1.3 | Hierarchy | 57 |
| 4.2 | The Main System | 58 |
| 4.3 | The Microcontroller | 60 |
| 4.3.1 | Clock and Reset Generator | 61 |
| 4.3.2 | Sequencer | 61 |
| 4.3.3 | Bus Arbiter | 65 |
| 4.3.4 | Memory Read and Write | 68 |
| 4.4 | The Dynamic Memory: DRAM | 72 |
| 4.5 | The Coprocessor | 74 |
| 4.5.1 | Clock Generator | 75 |
| 4.5.2 | Coprocessor Data Buffers | 76 |
| 4.6 | Summary | 77 |

| | |
|--|-----|
| 5. SHARED MEMORY CONFIGURATION | 81 |
| 5.1 Case Study | 82 |
| 5.2 Timing Characteristics | 85 |
| 5.2.1 Parameter Passing | 87 |
| 5.2.2 Bus Arbitration | 87 |
| 5.2.3 Busy-Wait Mechanism | 88 |
| 5.2.4 Interrupt Mechanism | 90 |
| 5.3 Relating Memory Accesses and Interface Mechanisms | 92 |
| 5.3.1 Varying Internal Operations and Memory Accesses | 94 |
| 5.3.2 Varying the Coprocessor Memory Access Rate | 96 |
| 5.3.3 Varying the Number of Coprocessor Memory Accesses | 98 |
| 5.4 Summary | 105 |
| 6. DUAL-PORT MEMORY CONFIGURATION | 107 |
| 6.1 General Description | 108 |
| 6.1.1 Contention Arbitration | 108 |
| 6.1.2 Read/Write Operations | 110 |
| 6.2 The System Architecture | 111 |
| 6.2.1 Dual-Port Memory Model | 111 |
| 6.2.2 The Coprocessor | 113 |
| 6.2.3 Bus Interface Controller | 113 |
| 6.2.4 Coprocessor Memory Controller | 114 |
| 6.2.5 The Main Controller | 117 |
| 6.3 Timing Characteristics | 118 |
| 6.3.1 Interface Mechanisms | 120 |
| 6.4 Performance Results | 121 |
| 6.4.1 Varying Internal Operations and Memory Accesses | 121 |
| 6.4.2 Varying the Memory Access Rate | 126 |
| 6.4.3 Varying the Number of Memory Accesses | 127 |
| 6.4.4 Speedup Achieved | 129 |
| 6.5 Summary | 130 |
| 7. CACHE MEMORY CONFIGURATION | 133 |
| 7.1 Memory Hierarchy Design | 134 |
| 7.1.1 General Principles | 134 |
| 7.1.2 Cache Memory | 135 |

| | | |
|-------|--|-----|
| 7.2 | System Organization | 138 |
| 7.2.1 | Cache Memory Model | 138 |
| 7.2.2 | The Coprocessor | 139 |
| 7.2.3 | Coprocessor Memory Controller | 140 |
| 7.2.4 | The Bus Interface Controller | 145 |
| 7.3 | Timing Characteristics | 150 |
| 7.3.1 | Block Transfer During Handshake Completion | 155 |
| 7.4 | Performance Results | 158 |
| 7.4.1 | Varying the Number of Addressed Locations | 159 |
| 7.4.2 | Varying the Block Size | 162 |
| 7.4.3 | Varying the Number of Memory Accesses | 164 |
| 7.4.4 | Speedup Achieved | 166 |
| 7.4.5 | Miss Rate with Random Address Locations | 167 |
| 7.5 | Summary | 169 |
| 8. | ADVANCED TOPICS AND FURTHER RESEARCH | 173 |
| 8.1 | Conclusions and Achievements | 173 |
| 8.2 | Advanced Topics and Further Research | 176 |
| 8.2.1 | Complete VHDL Model | 176 |
| 8.2.2 | Cost Evaluation | 177 |
| 8.2.3 | New Configurations | 177 |
| 8.2.4 | Interface Synthesis | 177 |
| 8.2.5 | Architecture Synthesis | 177 |
| 8.2.6 | Framework for co-design | 177 |
| 8.2.7 | General Formalization | 178 |
| | Appendices | 185 |
| A | Benchmark Programs | 185 |
| B | Top-Level VHDL Model of the Co-design System | 191 |
| C | Translating PALASM TM into VHDL | 199 |
| D | VHDL Version of the Case Study | 205 |
| | References | 219 |
| | Index | 225 |

List of Figures

| | | |
|------|---|----|
| 2.1 | The co-design flow | 11 |
| 2.2 | Typical CLB connections to adjacent lines | 19 |
| 2.3 | Intel FLEXlogic iFX780 configuration | 21 |
| 2.4 | Target architecture with parameter memory in the coprocessor | 24 |
| 2.5 | Target architecture with memory-mapped parameter registers | 25 |
| 2.6 | Target architecture using a general-purpose processor and ASICs | 26 |
| 3.1 | Development route | 31 |
| 3.2 | Hardware synthesis process | 34 |
| 3.3 | Run-time system | 37 |
| 3.4 | Run-time system and interfaces | 38 |
| 3.5 | Target architecture | 39 |
| 3.6 | Bus interface control register | 43 |
| 4.1 | Main system configuration | 59 |
| 4.2 | Coprocessor board components | 60 |
| 4.3 | Logic symbol for the microcontroller | 60 |
| 4.4 | VHDL model for the clock and reset generator | 62 |
| 4.5 | Writing into the coprocessor control register | 63 |
| 4.6 | The busy-wait model | 64 |
| 4.7 | The interrupt routine model | 65 |
| 4.8 | Completing the handshake, by negating <i>Ncopro_st</i> | 66 |
| 4.9 | Algorithmic state machine for the bus arbiter | 67 |
| 4.10 | Algorithmic state machine for memory read/write | 70 |

| | | |
|------|--|-----|
| 4.11 | Logic symbol for the DRAM_16 | 72 |
| 4.12 | Algorithmic state machine for the DRAM | 73 |
| 4.13 | Logic symbol of the coprocessor | 74 |
| 4.14 | Logic symbol of the coprocessor clock generator | 75 |
| 4.15 | Flowchart for the coprocessor clock generator | 75 |
| 4.16 | Logic symbol of the coprocessor data buffers | 76 |
| 4.17 | Flowchart for the coprocessor input buffer control | 77 |
| 4.18 | Flowchart for the coprocessor output buffer control | 78 |
| 5.1 | C program of <i>example</i> | 83 |
| 5.2 | Modified C program | 84 |
| 5.3 | Passing parameter table to the coprocessor | 86 |
| 5.4 | Sampling of the bus request input signal (Nbr) | 88 |
| 5.5 | Bus arbitration without contention | 89 |
| 5.6 | Bus arbitration when using busy-wait | 90 |
| 5.7 | Handshake completion when using busy-wait | 91 |
| 5.8 | End of coprocessor operation with interrupt | 92 |
| 5.9 | Handshake completion when using interrupt | 93 |
| 5.10 | Graphical representation for T_{bw} and T_{int} , in terms of iterations | 97 |
| 5.11 | Graphical representation for T_b and T_i , in terms of accesses | 99 |
| 5.12 | Relation between $Nmem_{fin}$ and mem_{fin} , for busy-wait | 103 |
| 6.1 | Logic symbol of the dual-port memory | 108 |
| 6.2 | Arbitration logic | 110 |
| 6.3 | Main system architecture | 111 |
| 6.4 | Coprocessor board for the dual-port configuration | 112 |
| 6.5 | Logic symbol of DRAM16 | 112 |
| 6.6 | State machine for the dual-port memory model | 114 |
| 6.7 | Logic symbol of the coprocessor for the dual-port configuration | 115 |
| 6.8 | Logic symbol of the bus interface controller | 115 |
| 6.9 | Logic symbol of the coprocessor memory controller | 116 |
| 6.10 | State machine of the coprocessor memory accesses controller | 117 |
| 6.11 | State machine of the DRAM controller | 118 |
| 6.12 | Logic symbol of the main controller | 119 |

| | | |
|------|--|-----|
| 6.13 | Chart for T_b and T_i , in terms of <i>iterations</i> | 123 |
| 6.14 | Coprocessor memory access | 124 |
| 6.15 | Synchronization between memory controller and accelerator | 125 |
| 7.1 | Addressing different memory levels | 135 |
| 7.2 | Block placement policies, with 8 cache blocks and 32 main memory blocks | 136 |
| 7.3 | Coprocessor board for the cache memory configuration | 138 |
| 7.4 | Logic symbol of the cache memory model | 139 |
| 7.5 | Flowchart for the cache memory model | 140 |
| 7.6 | Logic symbol of the coprocessor for the cache configuration | 140 |
| 7.7 | Logic symbol for the coprocessor memory controller | 141 |
| 7.8 | Virtual address for the cache memory configuration | 141 |
| 7.9 | Cache directory | 142 |
| 7.10 | Algorithmic state machine for the coprocessor memory controller | 143 |
| 7.11 | Logic symbol of the bus interface controller for the cache configuration | 146 |
| 7.12 | Algorithmic state machine for the block transfer | 147 |
| 7.13 | Algorithmic state machine for the block updating | 149 |
| 7.14 | Coprocessor cache memory write | 151 |
| 7.15 | Bus arbitration when there is a cache miss | 152 |
| 7.16 | Transferring a word from the cache to the main memory | 153 |
| 7.17 | End of the block transfer during a cache miss | 155 |
| 7.18 | End of coprocessor operation and beginning of block update | 157 |
| 7.19 | End of transfer of block 0 | 158 |
| 7.20 | Handshake completion after block updating | 159 |
| 7.21 | C program example, with random address locations | 168 |

List of Tables

| | | |
|-----|--|-----|
| 2.1 | Some characteristics of the XC4000 family of FPGAs | 18 |
| 3.1 | Performance results for PLUM | 47 |
| 3.2 | Performance results for EGCHECK | 48 |
| 5.1 | Performing 10 internal operations and a single memory access per iteration | 94 |
| 5.2 | Performing 100 internal operations and one memory access per iteration | 96 |
| 5.3 | Performing 10 iterations and a single memory access per iteration | 96 |
| 5.4 | Performing 10 iterations and 10 internal operations | 98 |
| 5.5 | Performing 10 internal operations and no memory accesses | 100 |
| 5.6 | Performing 10 internal operations and 2 memory accesses per iteration | 101 |
| 5.7 | Performing 10 internal operations and 3 memory accesses per iteration | 101 |
| 6.1 | Performing 10 internal operations and a single memory access per iteration | 121 |
| 6.2 | Performing 10 internal operations and a single memory access per iteration | 126 |
| 6.3 | Performing 10 iterations and a single memory access per iteration | 127 |
| 6.4 | Performing 10 iterations and 10 internal operations | 128 |
| 6.5 | Performing 10 internal operations and 2 memory accesses per iteration | 129 |

| | | |
|-----|---|-----|
| 7.1 | Performing 10 operations and 1 memory access per iteration with $BS = 512$ bytes | 160 |
| 7.2 | Performing 10 operations and 1 memory access per iteration, with $BS = 256$ bytes | 163 |
| 7.3 | Performing 10 operations and 1 memory access per iteration, with $BS = 128$ bytes | 164 |
| 7.4 | Performing 10 iterations and 10 internal operations, with $BS = 512$ bytes | 165 |
| 7.5 | Performing 10 internal operations and 1 memory access per iteration, with $BS = 512$ bytes and random address locations | 167 |
| 7.6 | Performing 10 operations and 1 memory access per iteration, with $BS = 128$ bytes and random address locations | 169 |

Preface

In this Book, we are concerned with studying the co-design methodology, in general, and how to determine the more suitable interface mechanism in a co-design system, in particular. This will be based on the characteristics of the application and those of the target architecture of the system. We provide guidelines to support the designer's choice of the interface mechanism.

The content of this book is divided into 8 chapters, which will be described in the following:

In Chapter 2, we present co-design as a methodology for the integrated design of systems implemented using both hardware and software components. This includes high-level synthesis and the new technologies available for its implementation. Recent work in the co-design area is introduced.

In Chapter 3, the physical co-design system developed at UMIST is then presented. The development route adopted is discussed and the target architecture described. Performance results are then presented based on experimental results obtained. The relation between the execution times and the interface mechanisms is analysed.

In order to investigate the performance of the co-design system for different characteristics of the application and of the architecture, we developed, in Chapter 4, a VHDL model of our co-design system.

In Chapter 5, a case study example is presented, on which all the subsequent analysis will be carried out. The timing characteristics of the system are introduced, that is times for parameter passing and bus arbitration for each interface mechanism, together with their handshake completion times. The relation between the coprocessor memory accesses and the interface mechanisms is then studied.

In Chapter 6, a dual-port shared memory configuration is introduced, in substitution to the single-port shared memory of the original configuration. This new configuration aims to reduce the occurrence of bus contention, naturally present in a shared bus architecture. Our objective is to identify performance

improvements due to the substitution of the single-port shared memory by the dual-port shared memory.

In Chapter 7, A cache memory for the coprocessor is later on introduced into the original single-port shared memory configuration. This is an alternative to the dual-port memory, allowing us to reduce bus contention, while keeping the original shared memory configuration. The identification of performance improvements, due to the inclusion of a cache memory for the coprocessor in the original implementation, is then carried out.

In Chapter 8, we describe new trends in co-design and software acceleration.

N. NEDJAH AND L. M. MOURELLE

Acknowledgments

We are grateful to FAPERJ (Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro, <http://www.faperj.br>) and CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico, <http://www.cnpq.br>) for their continuous financial support.

Chapter 1

INTRODUCTION

In a digital system, hardware is usually considered to be those parts of the system implemented using electronic components, such as processors, registers, logic gates, memories and drivers. Software is thought of as the sub-systems implemented as programs stored in memory as a sequence of bits, which are read and executed by a processor. In a traditional design strategy, the hardware and software partitioning decisions are fixed at an early stage in the development cycle and both designs evolve separately (Kalavade and Lee, 1992; Kalavade and Lee, 1993; N. S. Woo and Wolf, 1994). Certain operations are clearly implemented by hardware, such as high-speed data packet manipulation; others by software, such as recursive search of a tree data structure; and there are usually a collection of further operations that can be implemented either by hardware or by software. The decision to implement an operation in hardware or software is based on the available technology, cost, size, maintainability, flexibility and, probably most importantly, performance.

Advances in microelectronics have offered us large systems containing a variety of interacting components, such as general-purpose processors, communications sub-systems, special-purpose processors (e.g., Digital Signal Processors – DSP), micro-programmed special-purpose architectures, off-the-shelf electronic components, logic-array structures (e.g., Field Programmable Gate Arrays – FPGAs) and custom logic devices (e.g., Application Specific Integrated Circuits – ASICs) (Subrahmanyam, 1992; Subrahmanyam, 1993; Wolf, 2004).

Today's products contain embedded systems implemented with hardware controlled by a large amount of software (G. Borriello, 1993). These systems have processors dedicated to specific functions and different degrees of

programmability (Micheli, 1993) (e.g., microcontrollers), namely the application, instruction or hardware levels. In the application level, the system is running dedicated software programs that allows the user to specify the desired functionality using a specialized language. At the instruction level, programming is achieved by executing on the hardware, the instructions supported by the architecture. hardware-level programming means configuring the hardware, after manufacturing, in the desired way. There have also been advances in some of the technologies related to design system, such as logic synthesis, and system level simulation environments, together with formal methods for design specification, design and verification (Subrahmanyam, 1992; Subrahmanyam, 1993).

In this chapter, we discuss the concept of synthesis and design approaches. Later on, the co-design concept is introduced, together with a methodology for its application. Finally, the objective of this book is then presented, together with its structure.

1.1 Synthesis

Synthesis is the automatic translation of a design description from one level of abstraction to a more detailed, lower level of abstraction. A behavioral description defines the mapping of a system's inputs to its outputs and a structural description indicates the set of interconnected components that realize the required system behavior. The synthesis process offers reduction in the overall design time and cost of a product, together with a guarantee of the circuit correctness. However, the synthesized design may not be as good as one produced manually by an experienced designer. Architectural synthesis allows the search for the optimal architectural solution relative to the specified constraints (E. Martin and Philippe, 1993).

High-level synthesis is the process of deriving hardware implementations for circuits from high-level programming languages or other high-level specifications (Amon and Borriello, 1991). In order to have a behavioral description of the system, we use a hardware Description Language (HDL), which offers the syntax and semantics needed. As examples of suitable HDLs, we have the Specification and Description Language (SDL) (Rossel and Kruse, 1993; A. A. Jerraya and Ismail, 1993), the Very high speed integrated circuits hardware Description Language (VHDL) (Ecker, 1993a; Ecker, 1993b; Navabi, 1998), the high-level description language *HardwareC* (Gupta, 1993; R. K. Gupta and Micheli, 1992a; R. K. Gupta and Micheli, 1992b; R. K. Gupta and Micheli, 1994; Gupta and Micheli, 1992; Gupta and Micheli, 1993; Ku and Micheli, 1990). Instead of a hardware description language, it is possible to use a high-level programming language, such as C or C++, to describe the system behavior, making the translation to a hardware description language at a later stage of the design process (M. D. Edwards, 1993; Edwards and Forrest, 1994; Edwards

and Forrest, 1995; Edwards and Forrest, 1996a; Edwards and Forrest, 1996b; P. Pochmuller and Longsen, 1993).

Using high-level synthesis techniques, we can synthesize digital circuits from a high-level specification (Thomas, 1990; D. E. Thomas and Schmit, 1993) which have the performance advantages offered by customizing the architecture to the algorithm (Srivastava and Brodersen, 1991; M. B. Srivastava and Brodersen, 1992). Nevertheless, as the number of gates increases, the cost and turnaround time, i.e., the time taken from the design specification until its physical implementation, also increase, and for large system designs, synthesized hardware solutions tend to be expensive (Gupta, 1993; Gupta and Micheli, 1993).

Despite the increase in power and flexibility achieved by new processors, users are always asking for something more. In order to satisfy this demand, it would be necessary to add special-purpose modules to a processor system. However, this would constrain the system and the usually small market generated for a specialized function discourages companies from developing hardware to satisfy such needs. A solution to this problem would be the use of a reconfigurable system, based on, say, Field-Programmable Gate Arrays (FPGAs), which could be attached to a standard computer, for performing functions normally implemented by special-purpose cards (D. E. Van den Bout, 1992). This kind of implementation offers faster execution, low cost and low power, since the necessary logic is embedded in the ASIC component.

1.2 Design Approaches

For a behavioral description of a system implemented as a program, running on a processor, we have a low cost and flexible solution. Typically, the pure software implementations of a system are often too slow to meet all of the performance constraints, which can be defined for the overall time (latency) to perform a given task or to achieve predetermined input/output rates (R. K. Gupta and Micheli, 1992b). Depending on these constraints, it may be better to have all or part of the behavioral description of the system implemented as a hardware circuit. In this kind of implementation, hardware modifications cannot be performed as dynamically as in a software implementation and the relative cost is higher, since the solution is customized.

System designers are faced with a major decision: what kind of implementation is the best, given a behavioral description of a system and a set of performance constraints – a software or a hardware solution? Cost-effective designs use a mixture of hardware and software to accomplish their overall goals (Gupta, 1993; Gupta and Micheli, 1993). Dedicated systems, with hardware and software tailored for the application, normally provide performance improvements over systems based on general-purpose hardware (Srivastava and Brodersen, 1991; M. B. Srivastava and Brodersen, 1992). Mixed system