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

To my mother and sisters,
Nadia

To my father (in memory)
and mother, Luiza

Contents

List of Figures

List of Tables

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, communicationsub-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 circuit s 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 circuit s 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