Preface

As CMOS semiconductor technology strides towards billions of transistors on a single die new problems arise on the way. They are concerned with the diminishing fabrication process features, which affect for example the gate-to-wire delay ratio. They manifest themselves in greater variations of size and operating parameters of devices, which put the overall reliability of systems at risk. And, most of all, they have tremendous impact on design productivity, where the costs of utilizing the growing silicon 'real estate' rocket to billions of dollars that have to be spent on design, verification, and testing. All such problems call for new design approaches and models for digital systems. Furthermore, new developments in non-CMOS technologies, such as single-electron transistors, rapid single-flux-quantum devices, quantum dot cells, molecular devices, etc., add extra demand for new research in system design methodologies.

What kind of models and design methodologies will be required to build systems in all these new technologies? Answering this question, even for each particular type of new technology generation, is not easy, especially because sometimes it is not even clear what kind of elementary devices are feasible there. This problem is of an interdisciplinary nature. It requires an bridges between different scientific communities. The bridges must be built very quickly, and be maximally flexible to accommodate changes taking place in a logarithmic timescale. On one side of such a bridge lies the nature (physics, chemistry, or biology) of the elementary devices that can be built in the chosen technology. On the other side are the functional properties of the desired computational operators. The recent developments in the area of DNA computing serve as a very good example of interdisciplinary bridge construction.

This book is, however, not about front-end nanotechnologies and novel computational mechanisms made possible on their basis. It is about one, relatively old and well-known behavioral paradigm in computing, called concurrency, and the ways concurrency is exhibited or can be exploited in relatively standard, digital hardware devices. Nevertheless, our hope is that the book will contribute to the series of publications aimed to be interdisciplinary. The above-mentioned bridge here should not be too challenging to construct as the 'banks of the river' are not too far apart. They can be roughly termed theoretical computer science and electronic system engineering. Understanding of most of this book by the representatives of both communities should not require additional background reading. So, thanks to this gap being not too wide, it seems possible to keep this material to a highly technical level, which should hopefully enable and encourage researchers standing on either side to start, or continue, thinking about new models, applications, design methods, algorithms, or software tools.

The book is aimed at inviting computer scientists working in the area of concurrency to look in the direction of hardware technology, of what is practically implementable regardless of theoretical bounds and restrictions. Remember Sean

Connery's saying in one of his numerious action movies: "It is impossible ... but it's doable!" This is exactly what engineers and practitioners often have in mind, when theoreticians tell them about computationally intractable problems or exponential growths.

On the other hand, computer scientists often go for solutions to real-world problems that are far too abstract and require layers of intermediate software packages to implement. A hardware engineer (not perhaps with a soldering iron these days!) can come up with an elegant and simple solution that can be realized in a handful of logic gates or transistors. For example, controlling concurrent processes and asynchronous interactions is relatively easy to implement in circuits. Yet, complicated mechanisms are required to be built on top of operating system kernels in software to support concurrency between different execution threads. In the end, what is clear and transparent mathematically becomes obscure and unpredicatble in realization.

This book also aims to give some messages to theoreticians, often relying on abstract computational models that are far from physical reality. As an example, many of us attended courses on data structures and algorithms, where we learnt that a quick sort is much faster than a bubble sort ($O(n \log n)$ vs. $O(n^2)$). That analysis was done under the assumption that moving one element from one location of the array to another location takes a constant time. However, a circuit designer implementing such algorithms in hardware would immediately disagree with such a complexity analysis. Today, wire delays are comparable to gate delays, and moving data across a chip takes a time proportional to the length of the move. For this reason, hardware designers would probably choose bubble sort as the best approach to sort an array stored in a one-dimensional space, since all moves are local (i.e., take a constant time). Instead, moves in a quick sort are global, thus taking time $O(n)$, and resulting in an overall complexity of $O(n^2 \log n)$. The question is, at which level of abstraction is a quick sort better than a bubble sort? Definitely not at the physical level.

For many years, practical digital circuits were built using global clocking. Although the clock was often not needed to perform the system's main function (barring, perhaps, hard real-time systems, but the real-time clock is another issue!), using the clock to pulse time explicitly, at the very-high-frequency (up to hundreds of megahertz) level, helped to construct complex circuits without worrying about races and hazards in gates and flip-flops. The age of global clocking is however coming to an end, whether designers welcome this fact or not. There are fundamental problems with distributing the signal of the gigahertz clock uniformly and reliably to every part of several-square-centimeter silicon die with devices of less than 100 nanometers. The irony of the situation is that while the clock used to be the bodyguard against hazards and unpredictable asynchrony, it now becomes its first enemy! With the increasing clock frequency the effects of parametric jitter, due to thermal fluctuations for example, that may cause synchronization failures, become unavoidable. New ways of building hardware, using distributed clocking, are being sought. One such paradigm is called Globally Asynchronous Locally Synchronous (GALS) systems. It is based on the idea of a compromise between two contradictory issues. One is the fact that at the chip level communications have to be asynchronous and not rely on one central timing engine. The other is the fact that the productivity crisis forces designers to maximize the reuse of existing hardware solutions, the so-called intellectual property (IP) cores. Since the latter have been typically designed for clocked systems, they are deemed to remain intact and thus, when placed in the 'hostile' asynchronous surrounding, must be protected by using special wrappers. Whether this GALS paradigm, somewhat of a half-measure, is here to stay with us for another decade or longer does not really matter, but there is a clear meassage from the semiconductor technology people that 'serious climatic changes on the face of the die' have approached us already.

It is worth emphasizing here, particularly for the computer science audience, that temperature is one of the biggest threats to system design these days. Part of this problem boomerangs back on the same old designer's friend, the clock. The clock is responsible for radiating at least 40 % of the thermal energy on modern microprocessor chips.

The problems with variations of temperature and other parameters, such as switching thresholds of transistors, either in operation or at fabrication, add to the motivation for searching for ways of living without the clock at all. And here we come back to the subject of this book. Indeed, we must now look at self-timed or asynchronous circuit design, which is often associated with the real, physical world of concurrency. The history of research in asynchronous hardware design has traditionally been a fertile field for research on concurrency and Petri nets. Many new ideas about modeling and the analysis of concurrent systems, and Petri nets in particular, grew out of the theory of asynchronous digital circuits. For example, the theory of speed-independent circuits by D.E. Muller and W.C. Bartky laid the foundation for the important concepts of feasible sequences, cumulative states, finite equivalence classes, confluence, semimodularity, and so on. Similarly, the theory and practice of digital hardware design have always been watchful to the appearance of new results in concurrency models and analytic tools, including Petri nets, CSP, CCS, BDD and symbolic traversal, partial-order reductions, etc. Likewise, with clocked hardware, the sequential model of a finite state machine played the role of the 'theoretical mentor' to keep logic design well-disciplined.

The subject of concurrency and hardware design brings to our mind the question of what is specific about this relation and how it is different, if at all, from the use of concurrency in software engineering. One, perhaps, controversial idea is the following, which is based on the notion of computation distribution.

Hardware, especially when clockless, is distributed by its nature. So, in hardware design, the original concept of the system specification gradually becomes more distributed and concurrent as it evolves from the specification to its implementation. Thus the hardware synthesis is the problem of distributing a global system's state between a set of local states, the states of registers and flip-flops. In software design the process is somewhat reversed. One often starts with a possibly ambitious object-oriented and highly concurrent model. In the end everything must eventually be run on a single processor, or even a set of parallel processors, where the computations are sequential by definition (unless and until they become concurrent again at the microcode level thanks to superscalar and asynchronous hardware!). The design process, seen as solving a synthesis problem, must therefore schedule concurrent actions in a sequential time line, collecting the local states into the global state of the C or machine-level program. Both of these views are closely related, especially in the world of embedded systems, increasingly implemented as Systems-on-a-Chip (SoCs), where the traditional understanding of the boundary between hardware and software becomes very blurred. Because of that, it is most natural to consider the area of designing embedded systems also within the scope of interest of this volume.

In the last four years we have been involved in several attempts to bring the theory of concurrency and Petri nets closer to hardware design applications. These included two special workshops (1998, Lisbon, and 1999, Williamsburg) and one advanced tutorial (2000, Aarhus) held within the annual International Conference on Applications and Theory of Petri Nets. Two conferences on the Application of Concurrency to System Design have been held (1998, Auzu, Japan, and 2002, Newcastle, UK). One collective monograph, *Hardware Design and Petri Nets*, was published in 2000.

The papers collected in this book cover the scope of applications of concurrency techniques in hardware design. They are organized into four parts. Part I deals with formal models of digital circuits using process-based and net-based descriptions, and demonstrates their interrelationship. The three contributions here cover the problem of process composition satisfying certain safety and progress properties, the problem of the decomposition of signal transition graphs for more efficient synthesis of asynchronous circuits, and finally the method of translating delay-insensitive process algebra specification to Petri nets for their subsequent synthesis into circuits. Part II introduces approaches to designing asynchronous circuits. One of them is based on the idea of distributed clocking of arrays of processing elements, which nicely complements the micropipeline approach. The other presents new methods for the synthesis of asynchronous circuits using structural approximations and interface-preserving transformations on signal transition graphs. The papers in Part III deal with design support for embedded systems. One of them presents the functional model, the system architecture, and the mapping between functional blocks and architectural elements within VCC, a widely used tool for system-level design. The other paper looks at the design methodology for highly heterogenous embedded systems that is currently implemented in the new-generation tool Metropolis. Finally, Part IV addresses the problems of timed verification and performance analysis of hardware using symbolic techniques. The first contribution introduces a methodology for analyzing timed systems symbolically, from a formula characterizing sets of timed states, and applies it to timed guarded command models of circuits. The second paper presents a symbolic approach to analyzing the performance of asynchronous circuits using discrete-time Markov chains, and shows a method for battling the state-space explosion problem.

Acknowledgements. We are very much indebted to all the authors contributing to this book, and to David Kinniment, Alex Kondratyev, Maciej Koutny, Oded Maler, and Fei Xia, who acted as external referees (note that the papers were also cross-refereed internally), for their constructive criticism. Finally, we would like to mention the excellent cooperation with Springer-Verlag during the preparation of this volume.

September 2002

Jordi Cortadella
Alex Yakovlev
Grzegorz Rozenberg