Preface

How do we design efficient digital machines? Software programmers would say "by writing better code". Hardware designers would say "by building faster hardware". This book is on codesign – the practice of taking the best from software design and the best from hardware design to solve design problems. Hardware/software codesign can help a designer to make trade-offs between the flexibility and the performance of a digital system. Using hardware/software codesign, designers are able to combine two radically different ways of design: the sequential way of decomposition in time, using software, with the parallel way of decomposition in space, using hardware.

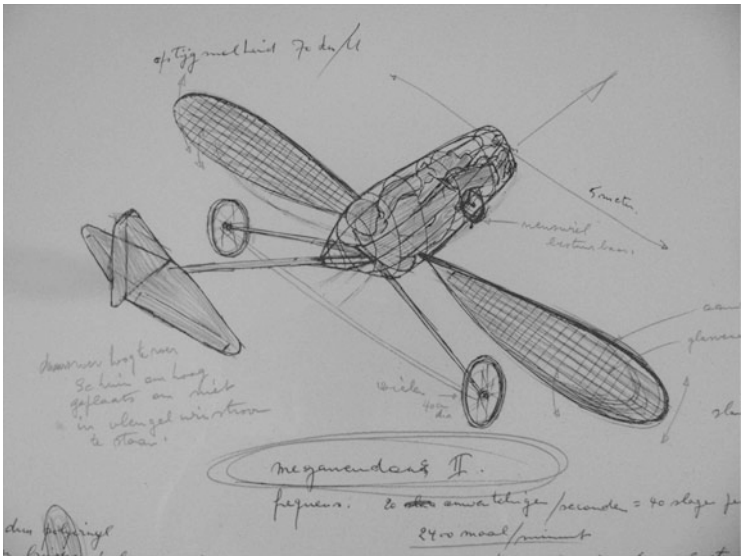
About the Picture

The picture on the next page is a drawing by a famous Belgian artist, Panamarenko. It shows a human-powered flying machine called the Meganeudon II. He created it in 1973. While, in my understanding, noone has built a working Meganeudon, I believe this piece of art captures the essence of design. Design is not about complexity, and it is not about low-level details. Design is about ideas, concepts, and vision. Design is a fundamentally creative process.

But to realize a design, we need technology. We need to map ideas and drawings into implementations. Computer engineers are in a privileged position. They have the background to convert design ideas into practical realizations. They can turn dreams into reality.

Intended Audience

This book assumes that you have a basic understanding of hardware, that you are familiar with standard digital hardware components such as registers, logic gates,



Panamarenko's Meganeudon II ((c) Panamarenko)

and components such as multiplexers, and arithmetic operators. The book also assumes that you know how to write a program in C. These topics are usually covered in an introductory course on computer engineering, or in a combination of courses on digital design and software engineering.

The book is suited for advanced undergraduate students and beginning graduate students, as well as researchers from other (non-computer engineering) fields. For example, I often work with cryptographers who have no formal training in hardware design but still are interested in creating dedicated architectures for highly specialized algorithms. This book is also for them.

Organization

The book puts equal emphasis on design methods, and modeling (design languages). Design modeling helps a designer to think about a design problem, and to capture a solution for the problem. Design methods are systematic transformations that convert design models into implementations.

There are four parts in this book: Basic Concepts, the Design Space of Custom Architectures, Hardware/Software Interfaces, and Applications.

Part I: Basic Concepts

Chapter 1 covers the fundamental properties of hardware and software, and discusses the motivation for hardware/software codesign. Chapters 2 and 3 describe data-flow modeling and implementation. Data-flow modeling is a system-level specification technique, and a very useful one. Data-flow models are implementation-agnostic: they map into software as well as into hardware. They also support high-level performance analysis and optimization. Chapter 2 in particular discusses stability analysis, and optimizations such as pipelining and retiming. Chapter 3 shows how dataflow models can be realized in hardware and software. Chapter 4 introduces control-flow and data-flow analysis of C programs. By analyzing the control dependencies and the data dependencies of a C program, a designer obtains insight into possible hardware implementations of that C program.

Part II: The Design Space of Custom Architectures

The second part is a tour along the vast design space of flexible, customized architectures. A review of four digital architectures shows how hardware gradually evolves into software. The Finite State Machine with Datapath (FSMD) discussed in Chap. 5 is the starting point. FSMD models are the equivalent of hardware modeling at the register-transfer level (RTL). Chapter 6 introduces micro-programmed architectures. These are still very much like RTL machines, but they have a flexible controller, which allows them to be reprogrammed with software. Chapter 7 reviews general-purpose embedded RISC cores. These processors are the heart of typical contemporary hardware/software systems. Finally, Chap. 8 ties the general-purpose embedded core back to the FSMD in the context of a System-on-Chip architecture (SoC). The SoC sets the stage for the hardware/software codesign problems that are addressed in the third part.

Part III: Hardware/Software Interfaces

The third part describes the link between hardware and software in the SoC architecture, in four chapters. Chapter 9 introduces the key concepts of hardware/software communication. It explains the concept of synchronization schemes and the difference between communication-constrained design and computation-constrained design. Chapter 10 discusses on-chip bus structures and the techniques they use to move information efficiently between hardware and software. Chapter 11 describes micro-processor interfaces. These interfaces are the locations in a processor-based design where custom-hardware modules can be attached. The chapter describes a memory-mapped interface, the coprocessor interface, and a custom-instruction

interface. Chapter 12 shows how hardware modules need to be encapsulated in order to "fit" into a micro-processor interface. This requires the design of a programmer's model for the custom hardware module.

Part IV: Applications

The final part describes three in-depth applications of hardware-software codesign. Chapter 13 presents the design of a coprocessor for the Trivium stream cipher algorithm. Chapter 14 presents a coprocessor for the Advanced Encryption Standard. Chapter 15 presents a coprocessor to compute CORDIC rotations. Each of these designs uses different processors and microprocessor interfaces. Chapter 13 uses an 8051 microcontroller and an ARM, Chap. 14 uses an ARM and a Nios-II, and Chap. 15 uses a Microblaze.

Many of the examples in this book can be downloaded. This supports the reader in experiments beyond the text. The Appendix contains a guideline to the installation of the GEZEL tools and the examples.

Each of the chapters includes a Problem Section and a Further Reading Section. The Problem Section helps the reader to build a deeper understanding of the material. Solutions for selected problems can be requested online through Springerextras (http://extras.springer.com).

There are several subjects which are *not* mentioned or discussed in this book. As an introductory discussion on a complex subject, I tried to find a balance between detail and complexity. For example, I did not include a discussion of advanced concepts in software concurrency, such as threads, and software architectures, such as operating systems and drivers. I also did not discuss software interrupts, or advanced system operation concepts such as Direct Memory Access.

I assume that the reader will go through all the chapters in sequence. A minimal introduction to hardware-software codesign should include Chaps. 1, 4, 5, 7–12.

A Note on the Second Edition

This book is the second edition of *A Practical Introduction to Hardware/Software Codesign*. The book was thoroughly revised over the first edition. Several chapters were rewritten, and new material was added. I focused on improving the overall structure, making it more logical and smooth. I also added more examples. Although the book grew in size, I did not extend its scope .

Here are some of the specific changes:

- The chapter on dataflow was split in two: one chapter on dataflow analysis and transformations and a second chapter on dataflow implementation. The

discussion on transformations offers the opportunity to introduce performance analysis and optimization early on in the book.
- Chapter 6 includes a new example on microcontroller-based microprogramming, using an 8051.
- Chapter 7, on RISC processors, was reorganized with additional emphasis on the use of the GNU Compiler Toolchain, inspection of object code, and analysis of assembly code.
- Chapter 8, on SoC, includes a new example using an AVR microcontroller. Support for the AVR instruction-set simulator was recently added to GEZEL.
- Part III, on Hardware/Software Interfaces, was reorganized. Chapter 9 explains the generic concepts in hardware/software interface design. In the first edition, these were scattered across several chapters. By bringing them together in a single chapter, I hope to give a more concise definition of the problem.
- Part III makes a thorough discussion of three components in a hardware/software interface. The three components are on-chip buses (Chap. 10), Microprocessor Interfaces (Chap. 11), and Hardware Interfaces (Chap. 12). "Hardware Interface" was called "Control Shell" in the first edition. The new term seems more logical considering the overall discussion of the Hardware/Software Interface.
- Chapter 10, On-chip Busses, now also includes a discussion on the Avalon on-chip bus by Alterea. The material on AMBA was upgraded to the latest AMBA specification (v4).
- Chapter 11, Microprocessor Interfaces, now includes a discussion of the NiosII custom-instruction interface, as well as an example of it.
- Part IV, Applications, was extended with a new chapter on the design of an AES coprocessor. The Applications now include three different chapters: Trivium, AES, and CORDIC.
- A new Appendix discusses the installation and use of GEZEL tools. The examples from Chapters 5, 6, 8, 11, 13–15 are now available in source code distribution, and they can be compiled and run using the GEZEL tools. The Appendix shows how.
- The extras section of Springer includes the solution for selected Problems.
- I did a thorough revision of grammar and correction of typos. I am grateful for the errata pointed out on the first edition by Gilberta Fernandes Marchioro, Ingrid Verbauwhede, Soyfan, and Li Xin.

Making it Practical

This book emphasizes ideas and design methods, in combination with hands-on, practical experiments. The book therefore discusses detailed examples throughout the chapters, and a separate part (*Applications*) discusses the overall design process.

The hardware descriptions are made in GEZEL, an open-source cycle-accurate hardware modeling language. The GEZEL website, which distributes the tools, examples, and other documentation, is at

```
http://rijndael.ece.vt.edu/gezel2
```

Refer to Appendix A for download and installation instructions.

There are several reasons why I chose not to use a mainstream HDL such as VHDL, Verilog, or SystemC.

- A first reason is *reduced modeling overhead*. Although models are crucial for embedded system construction, detailed modeling issues often distract the readers' attention from the key issues. For example, modeling the clock signal in hardware requires a lot of additional effort and it is not essential when doing single-clock synchronous design (which covers the majority of digital hardware design today).
- A second reason is that GEZEL comes with *support for cosimulation* built in. GEZEL models can be cosimulated with different processor simulation models, including ARM, 8051, and AVR, among others. GEZEL includes a library-block modeling mechanism that enables one to define new cosimulation interfaces with other simulation engines.
- A third reason is *conciseness*. This is a practical book with many design examples. Listings are unavoidable, but they need to be short. Chapter 5 further illustrates the point of conciseness with a single design example each in GEZEL, VHDL, Verilog, and SystemC side-by-side.
- A fourth reason is the path to implementation. GEZEL models can be translated (automatically) to VHDL. These models can be synthesized using standard HDL logic synthesis tools.

I use the material in this book in a class on hardware/software codesign. The class hosts senior-level undergraduate students, as well as first-year graduate-level students. For the seniors, this class ties many different elements of computer engineering together: computer architectures, software engineering, hardware design, debugging, and testing. For the graduate students, it is a refresher and a starting point of their graduate researcher careers in computer engineering.

In the class on codesign, the GEZEL experiments connect to an FPGA backend (based on Xilinx/EDK or Altera/Quartus) and an FPGA prototyping kit. These experiments are implemented as homework. Modeling assignments in GEZEL alternate with integration assignments on FPGA. Through the use of the GEZEL backend support, students can even avoid writing VHDL code. At the end of the course, there is a "contest". The students receive a reference implementation in C that runs on their FPGA prototyping kit. They need to accelerate this reference as much as possible using codesign techniques.

Acknowledgments

I would like to express my sincere thanks to the many people that have contributed to this effort.

My family is the one constant in my life that makes the true difference. I am more than grateful for their patience, encouragement, and enthusiasm. I remain inspired by their values and their sincerity.

The ideas in this book were shaped by interacting with many outstanding engineers. Ingrid Verbauwhede, my Ph.D. advisor and professor at Katholieke Universiteit Leuven, has supported GEZEL, for research and education, from its very start. She was also among the first adopters of the book as a textbook. Jan Madsen, professor at Denmark Technical University, and Frank Vahid, professor at University of California, Irvine, have been exemplary educators to me for many years. Every discussion with them has been an inspiration to me.

After the first edition of this book, I exchanged ideas with many other people on teaching codesign. I would like to thank Jim Plusquellic (University of New Mexico), Edward Lee (University of California at Berkeley), Anand Raghunathan (Purdue University), and Axel Jantsch (Royal Institute of Technology Sweden). I thank Springer's Chuck Glaser for encouraging me to write the second edition of the book and for his many useful and insightful suggestions. I thank Grant Martin (Tensilica) for writing a review on this book.

Throughout the years of GEZEL development, there have been many users of the tool. These people have had the patience to carry on, and bring their codesign projects to a good end despite the many bugs they encountered. Here are just a few of them, in alphabetical order and based on my publication list at http://rijndael.ece.vt.edu/gezel2/publications.html: Aske Brekling, Herwin Chan, Doris Ching, Zhimin Chen, Junfeng Fan, Xu Guo, Srikrishna Iyer, Miroslav Knezevic, Boris Koepf, Bocheng Lai, Yusuke Matsuoka, Kazuo Sakiyama, Eric Simpson, Oreste Villa, Shenling Yang, Jingyao Zhang. I'm sure there are others, and I apologize if I missed anyone.

Finally, I wish to thank the students at Virginia Tech that took the codesign class (ECE 4530). Each year, I find myself learning so much from them. They continue to impress me with their results, their questions, their ideas. They're all engineers, but, I can tell, some of them are really artists.

I hope you enjoy this book and I truly wish this material helps you to go out and do some real design. I apologize for any mistakes left in the book – and of course I appreciate your feedback.

Blacksburg, VA, USA Patrick R. Schaumont