

Hardware Design for Software People in the SoC Era

**Hardware Design for Software People in the SoC Era
Verilog, VHDL, SystemC and Matlab®**

by Steven Watkins
Copyright 2005 Blue Pacific Computing

Acknowledgments

The following products are registered trademarks with their respective companies:

Matlab and Simulink from the Mathworks
Modelsim from Mentor Graphics
SystemC from Open SystemC
Synplify from Synplicity
Spartan, Virtex and ChipScope from Xilinx
Cyclone, Stratix and Signal Tap from Altera

Thanks to David Wilson of the UCSD Bookstore for his valued assistance.

Ordering Information

First Edition, Second Printing, November, 2005

Order additional copies from UCSD Bookstore:

[Http://bookstore.ucsd.edu](http://bookstore.ucsd.edu)
Phone: 800 520-7323
858 534-4557

UCSD ISBN: 0-1007-2076-5

Table of Contents

Chapter 1	Introduction to the Hardware Description Languages	1
	Target Audience	1
	How To Use This Book	4
	The Four Hardware Description Languages	9
	How To Choose a Hardware Description Language	10
Chapter 2	Hardware in Electronic Systems	13
	The Four Hardware Technologies	14
	How To Choose a Hardware Technology	16
	ASIC and FPGA Design Flow	18
Chapter 3	Overview of the Languages	21
Chapter 4	Modeling Styles	27
	Behavioral and Analog	27
	Structural	27
	RTL	28
Chapter 5	Verilog and SystemVerilog	37
Chapter 6	VHDL	51
Chapter 7	SystemC and C/C++	63
Chapter 8	Matlab	71
Chapter 9	Basic Blocks and State Machines	73
Chapter 10	Logic Synthesis	93
	ASIC Logic Synthesis	93
	FPGA Logic Synthesis	94
Chapter 11	Verification, Manufacturing and Test	99
	Verification	100
	Design for Manufacturing	102
	Design for Test	103
Chapter 12	Design Reuse	107
	FPGA to ASIC Conversion	108
Chapter 13	Analog Modeling	109
Chapter 14	FPGA Design and Programming	111
	FPGA Tutorial and User Guide	118
Chapter 15	Design Software	143
	Simulator User Guide	145
Chapter 16	Labs	155
	Solutions	181
Appendix	Websites, Books, Acronyms and Reserved Words	279
Index		291

Blank Page

Chapter 1: Introduction to the Hardware Description Languages

Target Audience

This book was written for software engineers and managers who want to understand the basics of Electronic System Level (ESL) hardware design in the System-on-a-Chip (SoC) era. The book covers all the current approaches for using software to design electronic systems composed of integrated circuits. It is also intended to be an overview for digital designers who want to know more about other Hardware Description Languages. It provides a practical overview of all aspects of electronic system design, and should be helpful for managers and engineers struggling with global resource issues (such as managing outsourcing or remaining employed in the wake of outsourcing).

The languages used to design electronic system hardware are the Hardware Description Languages (HDL's). There are currently four main HDL's:

- Verilog and SystemVerilog
- VHDL
- SystemC and C/C++
- Matlab®

VHDL and Verilog are the most common HDL's. SystemVerilog is an extension to Verilog. SystemC is the most popular version of a C/C++ HDL. Matlab is a system simulation language, which people have started using as an HDL. SystemVerilog, SystemC and Matlab are sometimes called Electronic System Level (ESL) languages.

There are also four main hardware technologies being used today to implement electronic systems:

- Application Specific Integrated Circuits (ASIC's)
- Programmable Logic Devices including Field Programmable Gate Arrays (FPGA's)
- Structured ASIC's
- Platforms and Application Specific Standard Products (ASSP's)

Hands-On Approach with Free Software

The book assumes that you are familiar with at least one high-level language like C or C++, and then builds on that experience to explain modern hardware design using the Hardware Description Languages. The book uses a hands-on approach with labs and free software available from the Web.

While all four languages provide the same basic capability of using software to describe, simulate and create hardware, there are some significant differences among the languages. The book suggests which language or languages are best for you, based on your particular circumstances. The emphasis is on the software, and on how to do hardware design from the perspective of a software person. These days, you don't need to know about transistors to do hardware design. Somebody still has to worry about the transistors, probably just not you.

Author's Background

I have written this book because I enjoy teaching and crafting the simplest possible explanations of technical concepts. This is because I came to engineering in a roundabout way, and have spent a lot of time looking for clear, simple explanations of technology. As an undergraduate I studied physics and philosophy at the California Institute of Technology, where some of my best education occurred outside the classroom in discussions with Richard Feynman (a Nobel Laureate in physics who also happened to be an amazing conga drummer). I took one computer class in Fortran programming that involved using a punch card machine in a dank sub-basement of an engineering building, and decided that computers were not for me.

After graduation from Caltech, I took a path completely opposite of science and engineering. I became a professional musician, and played guitar in rock, folk and blues bands in the U.S. and Europe for five years. About the time I was ready for something new, some Caltech friends offered me the chance to learn electronics on the job at their startup in Silicon Valley. I discovered the fun of playing with microprocessors while working with Intel, Zilog, Motorola, AMD, National, DEC and Bell Labs. After working for a while, I attended graduate school at UCSD for a more formal background in computers and electrical engineering, and I received an M.S. in Computer Science and a Ph.D. in Electrical and Computer Engineering.

I have spent the last decade doing digital and mixed-signal integrated circuit design with schematic capture and the HDL's. I have also written a fair amount of software (mostly in C/C++ for Electronic Design Automation tools). My industry experience includes large semiconductor companies and small wireless startups. Along the way I have taught VHDL, Verilog, SystemC and C/C++ for UCSD, Synopsys, Cadence, TI, HP, National Semiconductor, Alcatel, Nokia, ARM and others.

Being an older graduate student has made me greatly appreciate clear explanations, and that is what I have tried to give you with this book. I have also tried to follow the spirit of *The C Programming Language* by Brian Kernighan and Dennis Ritchie, which was a life-altering book for me. I hope this book leads you down an educational and fun path.

Benefits of Software-Based Hardware Design

Our lives have been greatly transformed by the transistor, the integrated circuit and the digital computer. As a student of history, I first want to acknowledge some key historical figures: Bardeen, Brattain and Shockley, who invented the transistor in the late 1940's; Kilby and Noyce, who invented the integrated circuit in the late 1950's; and some of the fathers of the modern digital computer: Leibnitz in the 1600's, Babbage in the 1800's, Turing and von Neumann in the 1940's.

Digital computers can do five things well: organize data, search data, solve mathematical problems, control and communicate. Many people, including me, have also tried to get them to do a sixth thing: mimic human intelligence. This hasn't worked out too well so far, but that's another story. These five tasks that computers do well can be done with little or no additional hardware, beyond a general-purpose computer. But you can greatly increase the performance of these five tasks if you create some additional custom hardware.

Using software to design hardware is a means to implement your software algorithms in hardware, to create custom hardware, and reap the big benefit of increased system performance. Most of you will do this with Field Programmable Gate Arrays (FPGA's). Some of you will do this with Application Specific Integrated Circuits (ASIC's) or structured ASIC's. Some of you may use a general-purpose platform or Application Specific Standard Product (ASSP) that is customized mostly through software.

Electronic System Level Design

There are also some additional benefits beyond increased system performance to be gained by using HDL's. These additional benefits are based on the process of designing and simulating your software together with the hardware, and this process is called Electronic System Level (ESL) design. Once you have some experience with ESL design, you will be able to create a more robust electronic system with fewer bugs, and you will be able to get your system to market faster. ESL design promotes the creation of an executable specification in a high-level language such as C/C++, and this can unify your entire system design flow. By linking hardware and software development with this executable specification, the ambiguity of a non-executable text specification can be eliminated. The idea is to use exact hardware and software specifications for the input of the process, and see a working system as the output of the process. I want to promote ESL design, but I don't want to paint a picture of the current state of ESL design as a smooth highway to success. The road is still rather bumpy. See the summary at the end of this chapter for the balanced picture.

ESL design is also referred to as: hardware/software co-design, co-simulation and co-verification. SystemC and other C/C++ language subsets are most commonly used in true ESL design. Matlab, VHDL and SystemVerilog can play a part in the ESL design flow, but not in the co-simulation role, since most system software is written in C and C++.

How To Use This Book

This book emphasizes a hands-on approach using lab exercises and free design simulation software available on the Web. A software person's eyes often begin to glaze over when a hardware person starts talking about hardware. Using a simulator can be a fun learning experience, and this should minimize eye glazing.

If you are a manager, you should read only the first three chapters, and then skim the rest of the book to get an overview of Electronic System Level design.

If you are a software person and want to learn how to use an HDL, do the following:

- Read the first four chapters
- Pick a language
- Skim the chapter on that language (either Chapter 5, 6 or 7)
- Use Chapter 15 to install the design software on your computer
- Use the HDL labs in Chapter 16 to start writing code and simulating with your language of choice
- You may also want to use design examples from the Web listed in the Appendix
- Read additional chapters of interest

If you are a system person currently using Matlab, follow the above advice for a software person. Pick VHDL or Verilog, since using Matlab as an HDL involves translating Matlab code or Simulink blocks into VHDL or Verilog. The translation tools are imperfect, as they suffer from the artificial intelligence problem of not being as smart as you are. It helps to know what human generated code looks like.

The book also includes one chapter on FPGA Design and Programming, Chapter 14. While FPGA's represent only one of the four classes of hardware technology, they are the ultimate hardware for software people. The FPGA manufacturers make it feasible for you to run through the entire FPGA design and implementation process, by providing free design and synthesis tools and low-cost FPGA development boards (between \$100 and \$200). Chapter 14 contains step-by-step instructions on using a low-cost FPGA development board.

Note: words in italics represent reserved words used by one or more of the HDL's. The exceptions to this convention are the source code examples in the Solutions Section in Chapter 16 and the Reserved Words Section in the Appendix where no italics are used.

What is a Hardware Description Language?

To understand the basic concept of Hardware Description Languages all you need to know is that you start with a standard computer language like C, and then add two new elements: a means to represent concurrency (parallelism), and a simple interface called a module.

Since hardware usually does many things at once, the capability to specify concurrency is the key difference between HDL's and most other computer languages. The module represents an object-oriented interface into hardware blocks, allowing the details of the blocks to be hidden elsewhere.

Just to remind you of what can be represented by a typical computer language like C. There are seven basic concepts:

- Variables
- Data structures
- Assignment operators
- Logical and arithmetic operators
- Control flow constructs
- Subroutines
- Comments

Variables store information.

Data structures organize variables in special ways. Examples: arrays and records.

Assignment operators assign terms on the right side of an equation to the left side.

Logical and arithmetic operators perform the basic operations of: and, or, not, addition, subtraction, multiplication, division and so on.

Control flow is performed with *if*, *case*, *for*, *while* statements.

Subroutines allow a program to be structured with a main body that calls one or more layers of subroutines.

Comments are non-executable text in plain English to explain what the program is doing.

Example for Verilog

The seven basic concepts for Verilog:

Variables:	nets and registers
Data structures:	<i>array</i>
Assignment operators:	<i><=, =</i>
Logical and arithmetic operators:	<i>&, /, ~, +, -, *, /</i> etc.
Control flow:	<i>if, case, for, while</i>
Subroutines:	<i>task and function</i>
Comments:	<i>// This is a comment</i>

These are the two additional concepts for Verilog:

Concurrency:	<i>always @ (posedge clk)</i>
Module:	<i>module ha(a, b, sum, carry);</i>

Concurrency and Module

The two additional concepts of concurrency and module are represented in roughly the same way in all four HDL's. The concurrency statements look like function calls in Verilog, SystemVerilog and VHDL. They are function calls in C/C++, SystemC and Matlab. The main difference between a concurrency statement and a function call is that a concurrency statement cannot be used hierarchically. That is, one concurrency statement cannot call another concurrency statement.

Concurrency statements:

Verilog and SystemVerilog :	<i>always and initial</i> procedural statements
VHDL:	<i>process</i> statement
C/C++ and SystemC:	<i>process</i> function call
Matlab:	depends on the tool

The module interface statement always contains this information: the module name, its input and output ports. The module statement is used to build hierarchy: the lowest level modules will contain the actual workings of a design, while the higher level modules should only contain other modules.

Module statements:

Verilog and SystemVerilog:	<i>module</i> statement
VHDL:	<i>entity/architecture</i> pair statement
C/C++ and SystemC:	<i>module</i> statement
Matlab:	depends on the tool

See the example Verilog half adder below as an example of a simple lower level module. It specifies the module name, its inputs and outputs and also the two different logic gates which form a half adder (the xor gate is ^, the and gate is &).

```
module ha(a, b, sum, carry);           // module half adder
    input a, b;                       // inputs are a, b
    output sum, carry;                // outputs are sum, carry

    assign sum = a ^ b;               // sum = a xor b
    assign carry = a & b;             // carry = a and b
endmodule                             // end module
```

You might ask, where is the concurrency statement? The answer is: very simple designs don't need to use a concurrency statement, because concurrency is implied by the *assign* statement. But most designs are more complex and do use explicit concurrency statements (examples will be presented later in the book). Also, while the above half adder is a good first example, designers will normally work at a higher level of abstraction. So they will use a “+” operator to create an adder, rather than specifying the logic gates that underlie an adder.

The Main Idea: Software In and Hardware Out

The main idea of designing with HDL's is to create software that is debugged with digital simulation and waveform display tools. This software is then transformed into logic gates, registers and memory with synthesis tools, and finally mapped to transistors with place and route tools. The end result is a physical database that can be used to create an integrated circuit. This book will provide you with hands-on experience in the first step of software-based hardware design, using free digital simulation tools to create and debug your software.

What is Not Covered in the Book

While the book teaches you how to write synthesizable code, it does not lead you through any synthesis or place and route exercises, because these tasks are beyond the scope of this book. To become proficient at using synthesis and place and route tools, you need to take tool-specific classes or work with other engineers who are experienced with these tools. Some engineers tailor their entire careers to become experts at synthesis or place and route.

The book also does not cover the other back-end aspects of the hardware design flow such as LVS, DRC, formal verification, timing analysis and signal integrity. LVS stands for Layout Versus Schematic and is used to verify that the physical database matches the design as specified by the HDL code. DRC stands for Design Rule Checking and is used to make sure that the physical database does not violate any of the manufacturing rules for a particular chip fabrication process (rules such as limits to the width of metal lines). Formal verification is a form of verification based on abstract mathematics. Timing analysis involves checking the signal paths to make sure that the hardware will run at the desired clock frequency. Signal integrity tools verify that adjacent wires will not cause cross talk and interfere with each other.

The Four Hardware Description Languages: Verilog, VHDL, C/C++ and Matlab

You may be interested in learning one of the HDL's, but aren't sure which one is best for you. For many people the answer is usually simple: your boss will tell you. Usually, you don't have a choice and must use a particular HDL as dictated by your company.

For many digital designers HDL's are like religion. Designers will often swear by the first HDL they learn, and swear at any attempt to make them learn a different HDL. From my point of view, all HDL's are pretty much the same, but they do exhibit some differences. If you do have a choice, the following sections should help you make that choice.