

---

# **Real Chip Design and Verification Using Verilog and VHDL**

**Ben Cohen**



**VhdlCohen Publishing**  
**Los Angeles, California**  
<http://www.vhdlcohen.com/>

# Real Chip Design and Verification Using Verilog and VHDL

Published by:  
VhdlCohen Publishing  
P.O. 2362  
Palos Verdes Peninsula CA 90274-2362  
[vhdlcohen@aol.com](mailto:vhdlcohen@aol.com)  
<http://www.vhdlcohen.com>

---

Library of Congress Cataloging-in-Publication Data  
A C.I.P. Catalog record for this book is available from the Library of Congress

Real Chip Design and Verification Using Verilog and VHDL  
ISBN 0-9705394-2-8

---

Copyright © 2002 by VhdlCohen Publishing

All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without the prior written permission from the author, except for the inclusion of brief quotations in a review.

Printed on acid-free paper

Printed in the United States of America

# Contents

<b>FOREWORD CADENCE DESIGN SYSTEMS .....</b>	<b>IX</b>
<b>FOREWORD SYNPLICITY, INC. ....</b>	<b>XI</b>
<b>PREFACE .....</b>	<b>XIII</b>
<b>ABOUT THE DISK.....</b>	<b>XVII</b>
<b>ACKNOWLEDGEMENTS.....</b>	<b>XXV</b>
<b>ABOUT THE AUTHOR.....</b>	<b>XXVII</b>
<b>DISCLAIMER.....</b>	<b>XXVIII</b>
<b>1. OVERVIEW</b>	<b>1</b>
1.1 <b>Architecture Decomposition Process</b>	<b>2</b>
1.2 <b>Digital Architectural Classes</b>	<b>3</b>
1.3 <b>Digital Hardware Classes</b>	<b>4</b>
1.4 <b>Transitioning Requirements into Architecture</b>	<b>5</b>
<b>2. FUNDAMENTALS</b>	<b>7</b>
2.1 <b>Flip-Flop</b>	<b>8</b>
2.1.1    Synchronous Flip-Flop	8
2.2 <b>Latch</b>	<b>9</b>
2.3 <b>Synchronous Edge detect / One shot</b>	<b>11</b>
2.4 <b>Using Both Edges of a Clock</b>	<b>16</b>
2.4.1    ClockBoost	17
2.4.2    Negative Edge and Positive Edge Flip-Flop	17
2.5 <b>Registers</b>	<b>20</b>
2.5.1    Serial to Parallel	20

2.5.2	Parallel to Serial	22
2.5.3	Register File	24
<b>2.6</b>	<b>Counters</b>	<b>28</b>
2.6.1	Up/Down Binary Counter	28
2.6.2	Loadable Down-Counter	32
2.6.3	Building a free-running downcounter	35
2.6.4	Simple 5-bit Up-Counter with Terminal Count	37
2.6.5	LFSR Terminal Counters	39
2.6.6	One-Hot / One Cold Counters	46
2.6.7	Gray-Code	51
2.6.7.1	Designing Gray-Code Counters	52
2.6.7.2	Gray To Binary Conversion	53
2.6.7.3	Binary To Gray Conversion	56
2.6.8	Gray-Code Counter Implementation	57
2.6.9	Johnson	64
<b>2.7</b>	<b>Memories</b>	<b>68</b>
2.7.1	ROM	68
2.7.1.1	Application of Trigonometric Functions	72
2.7.2	RAM	76
2.7.2.1	Inferring Virtex or Virtex-E Block SelectRAM+	78
2.7.3	EDAC	79
2.7.4	First-In-First-Out Memories (FIFO)	102
<b>2.8</b>	<b>USING Primitives OF ASICs and FPGAs</b>	<b>110</b>
2.8.1	Example Considering Cell Architecture	110
2.8.2	Register Loading	116
2.8.3	Pipelining	117
2.8.4	Registering Outputs	117
2.8.5	Evaluating Timing Early	117
<b>2.9</b>	<b><i>ClockBoost and ClockLock</i></b>	<b>118</b>
2.9.1	<i>ClockLock ClockBoost</i> Applications	121
2.9.1.1	Clock Multiplication and Division	121
2.9.1.2	Removing Board Delay	123
2.9.2	Conclusions	125
<b>3.</b>	<b>SYNCHRONOUS/ ASYNCHRONOUS/ CLOCKS / RESET</b>	<b>127</b>
<b>3.1</b>	<b>Asynchronous data into synchronous system</b>	<b>128</b>
3.1.1	Metastability Definitions	128
3.1.2	Metastable Condition Explained	129
3.1.2.1	MTBF in Single Stage Flip-Flop	131
3.1.2.2	Two-Stage Synchronization Circuit	133
3.1.3	Metastability in Bussed Signals	136
3.1.3.1	Data Stable for two Consecutive Clocks	138
<b>3.2</b>	<b>Gated-Clocks</b>	<b>142</b>
<b>3.3</b>	<b>Ripple Counter</b>	<b>143</b>

---

<b>3.4</b>	<b><i>Flancter</i>: Set Flag in One Clock Domain, Clear in Another</b>	<b>147</b>
3.4.1	Overview	147
3.4.2	How It Works	148
3.4.3	So What' s Wrong With It?	149
3.4.4	Applications of the <i>Flancter</i>	149
3.4.5	<i>Flancter</i> HDL Code	151
3.4.6	Variations on a <i>Flancter</i>	154
<b>3.5</b>	<b>FIFO</b>	<b>157</b>
3.5.1	Passing Data By FIFO Between Clock Domains	157
3.5.2	FIFO Full and FIFO Empty	158
3.5.3	FIFO Pointers - Implemented as Binary Counters	158
3.5.4	FIFO Pointers - Implemented as Gray-Code Counters	159
3.5.5	FIFO Design	159
3.5.5.2	FIFO Full and Empty	160
3.5.6	Aynchronous FIFO Model	162
<b>3.6</b>	<b>Trouble-Free Switching Between Clocks</b>	<b>184</b>
<b>4.</b>	<b>VERIFICATION</b>	<b>187</b>
<b>4.1</b>	<b>When Does Verification Start?</b>	<b>188</b>
<b>4.2</b>	<b>Transaction Based Verification</b>	<b>189</b>
<b>4.3</b>	<b>Transactions Definition and Sequencing Methods</b>	<b>190</b>
4.3.1	Transaction Based Driving Methods	191
<b>4.4</b>	<b>Verification Detailed Example</b>	<b>191</b>
4.4.1	The DUT Model	192
4.4.2	Verilog Testbench	194
<b>4.5</b>	<b>Forcing SiGNAL Errors</b>	<b>212</b>
4.5.1	Verilog Solution	213
4.5.2	VHDL Solution	215
4.5.2.1	Method of Operation	216
4.5.2.2	Application Example	217
<b>4.6</b>	<b>EDAC Verification</b>	<b>222</b>
4.6.1	VHDL EDAC Transaction-Based Verification	222
4.6.2	Verilog EDAC Transaction-Based Verification	234
<b>5.</b>	<b>CONTROL MACHINES</b>	<b>245</b>
<b>5.1</b>	<b>Architectural Concepts</b>	<b>246</b>
<b>5.2</b>	<b>Cpu Design</b>	<b>248</b>
5.2.1	Instruction Set Architecture (ISA) Format Guidelines	248
5.2.2	Architectural Implementation	250
5.2.2.1	Identify major cycle types	252

5.2.2.2	Identify the main hardware resources	254
5.2.2.3	Draw a high level architecture diagram with control style definition	255
5.2.2.4	Write pseudo-microcode	258
5.2.2.5	Write HDL and Synthesize (FSM design) -- Verilog	268
5.2.2.6	Write Testbench and Test – Verilog	274
5.2.2.7	Write HDL and Synthesize (FSM design) -- VHDL	282
5.2.2.8	Write Testbench and Test – VHDL	289
5.2.2.9	Write HDL and Synthesize (Microcode design) -- Verilog	298
<b>6.</b>	<b>ARITHMETIC MACHINES</b>	<b>305</b>
6.1	<b>VHDL Arithmetic</b>	<b>306</b>
6.1.1	VHDL Numeric Signed/Unsigned Operators	306
6.2	<b>Verilog Arithmetic</b>	<b>327</b>
6.2.1	Verilog 1364-1995 Arithmetic	327
6.2.1.1	Understanding Types and Numbers	328
6.2.1.2	Signed Operations with Unsigned Registers	331
6.2.2	Verilog 1364-2001 Arithmetic	334
<b>7.</b>	<b>MIXED SIMULATION AND SYNTHESIS</b>	<b>341</b>
7.1	<b>Mapping Data Types</b>	<b>342</b>
7.1.1	VHDL Generics	342
7.1.2	Verilog Parameters	343
7.1.3	Port Modes	343
7.1.4	VHDL Port Types	343
7.1.5	Verilog Port Types	344
7.1.6	Verilog States	344
7.2	<b>Importing verilog into VHDL</b>	<b>345</b>
7.3	<b>Importing VHDL Into Verilog</b>	<b>349</b>
7.3.1	Importing VHDL Into Verilog Without a Shell	350
7.3.2	Importing VHDL Into Verilog With a Shell	354
7.4	<b>Synthesis with Mixed Language Source Files</b>	<b>355</b>
<b>8.</b>	<b>MINIMIZING DESIGN ERRORS</b>	<b>357</b>
8.1	<b>Introduction</b>	<b>358</b>
8.2	<b>Design Process and Error Constraining</b>	<b>359</b>
8.2.1	Requirement Specification	359
8.2.2	Architectural Specification	360
8.2.3	Verification Plan	360
8.2.4	Design Implementation and Management	360
8.2.4.1	Art	360
8.2.4.2	Design Guidelines	361

---

8.2.4.3	Reuse	361
8.2.4.4	Tools	362
8.2.5	Design Verification	363
8.2.5.1	Transaction Based Stimulus Generation	364
8.2.5.2	Verifier / Code Coverage	364
8.2.6	Reviews	364
<b>8.3</b>	<b>Swatting at bugs – The Intel Pentium 4 Experience</b>	<b>365</b>
<b>8.4</b>	<b>Conclusions and Recommendations</b>	<b>366</b>
<b>9.</b>	<b>VERILOG /VHDL COMPARISONS AND GUIDELINES</b>	<b>367</b>
<b>9.1</b>	<b>Verilog Drivers / Resolution functions</b>	<b>372</b>
<b>9.2</b>	<b>VHDL Drivers / Resolution functions</b>	<b>373</b>
<b>9.3</b>	<b>simulation schedulers differences</b>	<b>374</b>
9.3.1	Initialization	374
9.3.2	Updating of Projected Waveforms	375
9.3.3	Blocking/Nonblocking	376
<b>9.4</b>	<b>Verilog Coding Style for VHDL Users</b>	<b>379</b>
9.4.1	Port Headers	379
9.4.2	Begin-End Pairs	379
9.4.3	Simple If-Else-Type Tests	380
9.4.4	One-Bit Wire Declaration	380
9.4.5	Procedural Block Naming	380
9.4.6	Local Variable Declarations	380
9.4.7	`define as Parameter	381
9.4.8	Logical Operations and Comparisons	381
9.4.9	For-Loops Versus Repeat	382
9.4.10	Clock Definition in Testbenches	383
9.4.11	Port Associations with Constants	383
9.4.12	Verilog “OTHERS” Equivalent	384
9.4.13	Verilog Tasks and Functions at End of Module	384
9.4.14	FSM Verilog Coding Style	384
<b>9.5</b>	<b>Verilog / VHDL Reflections</b>	<b>385</b>
<b>10.</b>	<b>INDEX</b>	<b>387</b>



## **FOREWORD** *Cadence Design Systems*

Verilog and VHDL have been with us now for almost 20 years. During that time, we have watched them grow from esoteric new ways of specifying testbench and design functionality to the mainstay in any complex design methodology. Despite the wide usage and proliferation of HDL-based design methodologies, there is still much to be learned in this area for many people, from new designers just starting-out with HDLs to experienced engineers looking for ways to improve their productivity.

This new book by Ben Cohen is an invaluable addition to the existing literature on chip design using the Verilog and VHDL hardware description languages. As Ben notes in his Preface, the purpose of his book is not to teach either HDL, as there are already several books on the market that do an excellent job of describing and teaching the languages. However, as Ben also notes, a sound understanding of the HDL, though a requirement, is not sufficient. Understanding the HDL alone will not make you an expert logic designer, any more than learning C or C++ will make you an expert computer programmer.

One of the things that make this book particularly important is that it doesn't focus on just Verilog or VHDL, but rather on actual design and simulation using examples from both languages. No one actually designs using two languages at the same time, but more and more designers find themselves using IP in a different language or integrating their design with another one written in different language. In addition to driving the need for tools that support both languages, this is also driving the need for designers to understand both Verilog and VHDL at least well enough to be able to debug modules written in either language.

This book concentrates on common classes of hardware architectures and design problems, and focuses on the process of transitioning design requirements into synthesizable HDL code. Using his extensive, wide-ranging experience in computer architecture and hardware design, as well as in his training and consulting work, Ben provides numerous examples of real-life designs illustrated with VHDL and Verilog code. This code is shown in a way that makes it easy for the reader to gain a greater understanding of the languages and how they compare. All code presented in the book is included on the companion CD, along with other information, such as application notes.

Ben also covers a critical aspect for any real-life testbench creation: the use of transaction-based verification techniques. Designs are too complicated to continue to validate them exclusively at the individual signal level. In order to both improve performance and ensure that the tests actually check intended behavior, designers need to create tests and verify results at the transaction level. The book includes a chapter covering on Verilog and VHDL transaction level testing while also referring to C++ based transaction level test tools such as the open-source *Testbuilder* (available at [www.testbuilder.net](http://www.testbuilder.net)).

Cadence Design Systems is proud that some of its leading digital verification products were used in the creation of this book. *HAL* was used for HDL analysis and lint checking, which provides a perfect static verification complement to the dynamic simulation featured in the rest of the book. The high-performance, mixed-language simulator *NC-Sim* was used for verifying all of the examples in the book and the results were shown using the *SimVision* GUI and debug environment.

This book is one of the best investments that a logic designer can make. We are certain that it will be of enormous value to all those involved in HDL-based chip design for years to come.

Rahul Razdan  
Corporate Vice President - Systems and Functional Verification.  
Cadence Design Systems, Inc.



# FOREWORD *Synplicity, Inc.*

*Ready, Fire, Aim!* A humorous cliché, but uncomfortably applicable to describe the experience of many of our young digital designers. Most learn by doing, thus leaving a trail of first attempts that is less than impressive.

There is a gap in the education and training of many new entrants to our profession. The preeminent method of learning the practical techniques for HDL based digital design is to learn by experience. The consequence is that many designers become best capable later in their career. For many designers, this occurs at a time when their design skills are becoming less utilized as they progress down any number of career paths, such as architect, manager, application engineer, etc., leaving core design behind.

For those eager to learn the practical aspects of design, they often are left 'on their own'. There is a void in mentoring in engineering today. The designer must seek out information from books, from colleagues, from poking and probing through existing code to see what is typically done in the industry.

Universities often bolster the foundations of engineering, mostly theoretical, sometimes practical, but typically lagging industry. Thus, the recent graduate starts their career playing catch-up. Industry text books (as opposed to Academic textbooks) attempt to remain current but often are specific to a small slice of the design process -- such as books on HDL language semantics for simulation not synthesis, or books on design applications that ignore implementation. The designer is left to put the pieces together to form a whole.

In *Real Chip Design and Verification Using Verilog and VHDL*, Ben Cohen bridges gaps. He bridges the gaps in a designer's knowledge, he covers the gaps left by other texts. The focus on this book is to learn by example. The readers starting point is the macro elements and their implementation. Ben takes these elements, simulates them, synthesizes them, and leaves the reader with the ability to do the same. He defers giving bias to Verilog or VHDL, but rather acknowledges that both are in use, and thus the reader may utilize either or both in their career.

Ben bridges simulation and synthesis, and this acknowledges that implementation and verification must both be done in design. He then extends the discussion to more complex design objects, and discusses architectural tradeoffs in these components. These discussions extend to their impact for targeting FPGAs and ASICs, their final destination.

Ben continues to be a pragmatic author. His topics and writing are very accessible and pertinent to the engineering community he reaches. At Synplicity, we have benefited from Ben's knowledge of language and tools in his role as customer and author. Now you can too.

Andrew Dauman  
Vice President, Corporate Applications Engineering  
Sunnyvale, CA  
October, 2001



---

# PREFACE

**Now that you know a hardware description language (HDL), where do you go from here?** As a VHDL trainer and consultant, I experienced that many engineers understand the HDL from a software viewpoint, but do not know how to approach design problems. There is a fallacy that HDL is the panacea to all design issues, and that synthesis tools will perform the magic of translating the HDL into hardware. The reality is that synthesis is just a tool to help in the implementation of what is described. It is necessary that the authors of the HDL code understand the hardware architecture and implications of what is described. A proper HDL description of an incorrect or improper architecture does not necessarily yield a correct or optimum design. The following quotation, from a contributor in *comp.lang.vhdl* newsgroup, expresses this point in an interesting manner: *HDL (e.g., VHDL or Verilog), like any other EE design tool, assumes that the user has a working knowledge of electronics and digital design. I know many who are poor designers before learning HDL and remain poor designers after taking an HDL class.*

*Real Chip Design and Verification Using Verilog and VHDL* addresses the practical and real aspects of logic design, processes, and verification. It incorporates a collection of FPGA and ASIC design practices, and uses Verilog and VHDL as a tool for expression of the desired architectures. This book is not intended to teach either HDL, as there are several books specifically geared toward teaching the languages. However, it provides various architectural design primitives, applications, and verification techniques, along with design methodologies and common practices.

Logic design is an art that is learned, and often relearned by designers. There are several common classes of design problems and several common classes of hardware architectures including synchronization logic, counters, controllers, arithmetic elements, and storage elements. This book addresses those classes of designs with practical examples to expose the reader to variations in styles and approaches. The architectural issues, design decomposition, and HDL code in both VHDL and Verilog are discussed and demonstrated. Transaction-based testbenches with error injection methodologies demonstrate, by example, design verification techniques. Models used for this verification task include a counter and an EDAC (error detection and correction) logic with a RAM.

This book is intended as a training book in conjunction with an HDL class as a means to demonstrate the transition of design requirements into an HDL design. Specifically, it demonstrates by example the following:

1. Styles of hardware architectures.
2. Logic design architectural decomposition process and the translation of the architectures into HDL.
3. HDL coding styles.
4. Verification techniques with HDL using transaction-based methodologies.

Cadence *NC-Sim* simulator and *HAL* analysis and lint checking tools were used because of their levels of efficiency, accuracy, and maturity. Cadence represents a vendor that is a leader in the EDA industry. Synplicity *Synplify Pro*® FPGA synthesis tool was also extensively used because Synplicity is recognized as a vendor of advanced, efficient, and easy to use synthesis tools targeted for FPGAs, and now ASICs. Even though these specific tools were used, almost all of the information is tool independent.

**Chapter 1** provides an overview of the architectural decomposition process, and presents the classes of hardware designs. **Chapter 2** presents fundamental architectural elements used in the construction of designs. These include flip-flops, latches, synchronous edge detector, application of both edges of the clock, registers, counter styles (*e.g.*, *Binary*, *One-Hot*, *Gray*, *LFSR*, *Johnson*), memories including ROM, RAM, FIFO, and Error Detection and Correction (EDAC) logic. A trigonometric function defined in C, but implemented in HDL as a ROM, is also demonstrated. This chapter also addresses the importance of understanding the cell primitives and FPGA architecture including the clocking features of ASICs and FPGAs. Topics on clocking schemes and phase lock loops are discussed. **Chapter 3** addresses the synchronous/asynchronous aspects of the real world, and methods to resolve those issues. Metastability is explained, MTBF calculations are defined, and solutions in the handling of metastability are presented. The design of an asynchronous FIFO is demonstrated. The topic of crossing clock domains is also presented. **Chapter 4** addresses the verification issue and presents through two examples the transaction-based verification methodology. The topic of forcing design errors is also demonstrated in those examples, including the verification of a loadable counter and an EDAC model for a thirty-two-bit wide memory. **Chapter 5** focuses on control machines and uses a very simple CPU design to demonstrate implementation methodologies with FSM and microprogrammed solutions. **Chapter 6** addresses arithmetic intensive machines. It explains the application of SIGNED and UNSIGNED types in HDL. Verilog 1995/2001 type issues are demonstrated. **Chapter 7** explains and demonstrates mixed mode simulations and synthesis. **Chapter 8** presents a discussion on minimizing design errors and addresses miscellaneous design issues. **Chapter 9** compares Verilog to VHDL to enable users of one discipline to understand the language differences and nuances of the other discipline. It also provides Verilog coding style guidelines for VHDL and Verilog users.

---

All HDL code described in the book is on a companion CD. All code was verified and simulated with *NC-Sim version v03.30.1*<sup>1</sup>. All synthesizable code was synthesized with *Synplify Pro*<sup>®</sup> *version 6.2.4*<sup>2</sup>. The CD also includes application notes and files of practical use that were collected over a period of several years. EMACS editor for Windows, along with VHDL and Verilog modes is on CD. The CD includes data sheets and additional information on Synplicity's product line, and excellent Cadence's Verilog reference and HDL simulation documentation.

This book is intended for:

1. **Engineers.** Book provides classes of architectural examples and decomposition into HDLs. Engineers are better at copying and improving upon what is done, than from starting from scratch. This book will provide a head start in these processes.
2. **Trainers.** This book provides the focus of an advanced hardware design class using HDLs. Emphasis is on architecture, processes, methodologies, and style.
3. **College students.** Book demonstrates the hardware architectural processes.

A list of Verilog books that are often recommended includes:

*Verilog HDL : A Guide to Digital Design and Synthesis*, Samir Palnitkar, 396 pages, Prentice Hall 1996, ISBN: 0134516753

*Verilog HDL Synthesis, A Practical Primer*, J. Bhasker, 236 pages, Star Galaxy Publishing, ISBN 0-9650391-5-3

*A Verilog HDL Primer, Second Edition*, J. Bhasker, 1999, Star Galaxy Publishing, ISBN 0-9650391-7-X.

*The Verilog Hardware Description Language, Fourth Edition*, Thomas, D . E . / Moorby, Philip R , 354 Pages, Kluwer Academic Publishers 1998, 354 Pages

For information about the new features of Verilog, I recommend the book *VERILOG 2001, A Guide to the New Features of the Verilog Hardware Description Language*, Stuart Sutherland, 2002, Kluwer Academic Publishers, ISBN 0-7923-7568-8

---

<sup>1</sup> Cadence Design Systems, Inc. <http://www.cadence.com/>

*NC-Sim* is available on several platforms including Win98, Win2000, WinNT4.0, Linux, Unix, HPPA.

For a guided tour of the Cadence VHDL and Verilog Desktop simulator please go to following page.

<http://www.orcad.com/product/simulation/hdlsim/>

<sup>2</sup> Synplicity <http://www.synplicity.com>

For VHDL, I second Janick Bergeron's recommendation<sup>3</sup> for the book *VHDL Coding Styles and Methodologies, 2nd Edition*, 1999, Kluwer Academic Publishers, ISBN 0-7923-8474-1.

Another highly recommended book for VHDL is *The Designer's Guide to VHDL, 2nd Edition*, Peter J. Ashenden, 740 pages, Morgan Kaufmann Publishers, ISBN 1558606912

For verification, the book *Writing testbenches: Functional Verification of HDL Models*, Janick Bergeron<sup>4</sup> Kluwer Academic Publishers, ISBN 0-7923-7766-4 is recognized as a standard.

---

<sup>3</sup> *Writing testbenches, Functional verification of HDL models*, Janick Bergeron, Kluwer Academic Publishers 2000

<sup>4</sup> <http://janick.bergeron.com/>

# About The CD

Table 1 summarizes the contents of the enclosed CD.

**Table 1 Contents of Enclosed CD**

## Chapter 2


MODEL	FIGURE	DESCRIPTION
<a href="#">ch2/ff_reset.v</a>	2.1-1	Verilog Resettable Flip-Flop
<a href="#">ch2/ff_reset.vhd</a>	2.1-2	VHDL Resettable Flip-Flop
<a href="#">ch2/latch.v</a>	2.2-1	Verilog Latch
<a href="#">ch2/latch.vhd</a>	2.2-2	VHDL Latch
<a href="#">ch2/oneshot.v</a>	2.3-2	Verilog Code for Edge-Detect and Testbench
<a href="#">ch2/oneshot.vhd</a>	2.3-2	VHDL Code for Edge-Detect and Testbench
<a href="#">ch2/negposff.v</a>	2.4.2-1	Using Negative Edge of Flip-flop to Clock Data, and using Positive edge of Clock to Output Data to System
<a href="#">ch2/ser2parallel.v</a>	2.5.1-1	Simple Serial to Parallel Converter with External Synchronization
<a href="#">ch2/ser2parallel.vhd</a>	2.5.1-2	Simple Serial to Parallel Converter with External Synchronization
<a href="#">ch2/parallel2ser.v</a>	2.5.2-1	Verilog Parallel to Serial Converter
<a href="#">ch2/parallel2ser.vhd</a>	2.5.2-2	VHDL Parallel to Serial Converter
<a href="#">ch2/regfile.v</a>	2.5.3-1	Register File Inference in Verilog
<a href="#">ch2/regfile.vhd</a>	2.5.3-2	Register File Inference in VHDL
<a href="#">ch2/counter.v</a>	2.6.1-1	Binary Verilog Counter
<a href="#">ch2/counter.vhd</a>	2.6.1-2	Binary VHDL Counter
<a href="#">ch2/counterun.vhd</a>	2.6.1-4	Counter Example with Numeric_Unsigned Package
<a href="#">ch2/dncounter.v</a>	2.6.2-1	Verilog Model of a Loadable Down-Counter
<a href="#">ch2/dncounter.vhd</a>	2.6.2-2	VHDL Model of a Loadable Down-Counter
<a href="#">ch2/freerundncntr.v</a>	2.6.3-2	Down-Counter Configured as a Free-Running Counter
<a href="#">ch2/ freerundncntr.vhd</a>	2.6.3-3	Down-Counter Configured as a Free-Running Counter
<a href="#">ch2/counter5.v</a>	2.6.4-1	Simple 5-Bit Up-Counter with Terminal Count




<a href="#">ch2/counter5.vhd</a>	2.6.4-2	Simple 5-Bit Up-Counter with Terminal Count
<a href="#">ch2/lfsr4.vhd</a> & <a href="#">ch2/TestLFSR_tb.vhd</a>	2.6.5-1	Simulation of Four-bit LFSR
<a href="#">ch2/lfsr4count.vhd</a>	2.6.5-1	Four-Bit LFSR counter that counts to 16 states
<a href="#">ch2/TestLFSR4_tb.vhd</a>	2.6.5-3	Model of the testbench for the LFSR terminal counter
<a href="#">ch2/lfsr4count.v</a>	2.6.5-5	Verilog Model for 4-bit LFSR Counter
<a href="#">ch2/lfsr4count_tb.v</a>	2.6.5-5	Verilog Testbench for 4-bit LFSR Counter
<a href="#">ch2/counter1hot.v</a>	2.6.6-1	Verilog Model of One-Hot Counter
<a href="#">ch2/counter1hot.vhd</a>	2.6.6-2	VHDL Model of One-Hot Counter
<a href="#">ch2/counter1hotsafe.v</a>	2.6.6-4	Verilog Model of Safe One-Hot Counter
<a href="#">ch2/counter1hotsafetb.v</a>	2.6.6-6	Verilog Model testbench for the safe One-Hot Counter
<a href="#">ch2/counter1hotsafe.vhd</a>	2.6.6-7	VHDL Model of Safe One-Hot Counter
<a href="#">ch2/gray2bin_bad.v</a>	2.6.7.2-2	Non-Working but Conceptually Correct Gray-to-Binary Verilog Model
<a href="#">ch2/gray2bin.vhd</a>	2.6.7.2-3	Working Gray-to-Binary VHDL Model
<a href="#">ch2/gray2bin.v</a>	2.6.7.2-4	Parameterized and correct gray-to-binary Verilog model
<a href="#">ch2/bin2gray.v</a>	2.6.7.3-1	Verilog Binay to Gray Converter
<a href="#">ch2/bin2gray.vhd</a>	2.6.7.3-2	VHDL Binay to Gray Converter
<a href="#">ch2/graycntr.v</a>	2.6.8-1	Verilog Gray Code Counter with binary output
<a href="#">ch2/graycntr_lookup.v</a>	2.6.8-10	Verilog Gray Code Counter lookup style
<a href="#">ch2/graycntr.vhd</a>	2.6.8-1	VHDL Gray Code Counter with binary output
<a href="#">ch2/graycntr_lookup.vhd</a>	2.6.8-9	VHDL Gray Code Counter lookup style
<a href="#">ch2/graycntr_tb.v</a>	2.6.8-5	Verilog Testbench for the Graycode Counter
<a href="#">ch2/graycntr_tb.vhd</a>	2.6.8-7	VHDL Testbench for the Graycode Counter
<a href="#">ch2/Johnson.v</a>	2.6.9-1	Verilog Johnson Counter with terminal count
<a href="#">ch2/Johnson.vhd</a>	2.6.9-4	VHDL Johnson Counter
<a href="#">ch2/rom2.v</a>	2.7.1-1	Verilog ROM Model
<a href="#">ch2/rom6.vhd</a>	2.7.1-5	VHDL ROM Model
<a href="#">ch2/rom6const.vhd</a>	2.7.1-6	VHDL ROM Implication with Constants
<a href="#">ch2/romgen/vhdlsin.c</a>	2.7.1.1-1	C program to generate the Verilog SINE ROM
<a href="#">ch2/romgen/vhdlsin.h</a>	2.7.1.1-2	C support file: vhdlsin.h





<a href="#">ch2/romgen/makefile</a>	2.7.1.1-3	C makefile
<a href="#">ch2/sinrom.v</a> <a href="#">ch2/romgen</a>	2.7.1.1-4	Generated SINE ROM Model with Angles in half Degrees
<a href="#">ch2/sinromtop.v</a>	2.7.1.1-5	Simple Test and Simulation Results of the SINE ROM Model
<a href="#">ch2/ram.v</a>	2.7.2-1	Verilog Synchronous RAM
<a href="#">ch2/ram.vhd</a>	2.7.2-2	VHDL Synchronous RAM
<a href="#">ch2/edac_pb.vhd</a>	2.7.3-2	VHDL EDAC Package Declaration and Body Written by European Space Agency
<a href="#">ch2/edac32_rtl.vhd</a>	2.7.3-3	Error Detection and Correction Block
<a href="#">ch2/mem.vhd</a>	2.7.3-4	Memory Model with Check Bits
<a href="#">ch2/system2.vhd</a>	2.7.3-5	Memory System Model with EDAC
<a href="#">ch2/edac32hamming.v</a>	2.7.3-7	Verilog EDAC 32-bit Hamming Code Converted from VHDL Package
<a href="#">ch2/ram40.v</a>	2.7.3-8	RAM Memory Model
<a href="#">ch2/sysmem.v</a>	2.7.3-9	System Model with the Memory and EDAC
<a href="#">ch2/fifo.vhd</a>	2.7.4-2	VHDL FOFO Model
<a href="#">ch2/fifo.v</a>	2.7.4-3	Verilog Synchronous Transmit Mode FIFO
<a href="#">ch2/test4syncreset.vhd</a>	2.8.1-5	VHDL Reduce-XOR logic with synchronous Reset
<a href="#">ch2/test4asyncreset.v</a>	2.8.1-6	Reduce-XOR logic with Asynchronous Reset
<a href="#">ch2/test4asyncreset.vhd</a>	2.8.1-8	Reduce-XOR logic with Asynchronous Reset
<a href="#">ch2/test4asyncreset.v</a>	2.8.1-9	Re-clocking of Reset for use as Asynchronous Circuit Reset
<a href="#">ch3/metastbl.v</a>	3.1.2.2-3	Synchronizing Flip-Flops
<a href="#">ch3/metastbl_vect.v</a>	3.1.3.1-1	Verilog Code and Testbench for Verifying Data Stable for two Consecutive Clocks
<a href="#">ch3/metastbl_vect.vhd</a>	3.1.3.1-4	VHDL Code for Verifying Data Stable for two Consecutive Clocks
<a href="#">ch3/metastbl_vect_tb.vhd</a>	3.1.3.1-5	VHDL Testbench Code for Verifying Data Stable for two Consecutive Clocks
<a href="#">Ch3/ff_bad_design.v</a>	3.2-2	Verilog Model of Gated Clock
<a href="#">ch3/ripple.vhd</a>	3.3-1	VHDL Ripple Counter
<a href="#">ch3/ripple_tb.vhd</a>	3.3-2	Ripple Counter Testbench in VHDL
<a href="#">ch3/Flancter.v</a>	3.4.5-1	Verilog Flancter Code

ch3/Flancter.vhd	3.4.5-2	VHDL Flancter Code
ch3/fifo_async.vhd	3.5.6-1	VHDL model of the FIFO
ch3/fifo_async_tb.vhd	3.5.6-2	3.5.6-2 Simple, Non-Exhaustive FIFO Testbench
ch3/fifo_async.v ch3/fifo_async2.v	3.5.6-6	Verilog Model of FIFO
ch3/fifo_async_tb.v	3.5.6-7	Simple, Non-Exhaustive Verilog Testbench
ch3/switchclk.v	3.6-2	Verilog Model, Trouble-Free Switching between Clocks
ch3/switchclk.vhd	3.6-3	VHDL Model, Trouble-Free Switching between Clocks
ch4/counter.v	4.4.1-1	Verilog Model of Loadable Counter
ch4/counter.vhd	4.4.1-2	VHDL Model of Loadable Counter
ch4/counterandtb2.v	4.4.2-7	Verilog Model of Testbench that Includes Verification Environment
ch4/counterandtb2.vhd	4.4.2-10	VHDL Testbench for Counter
ch4/a.vhd	4.5.2.2-1	Test Component
ch4/testforcetop.vhd	4.5.2.2-2	Testbench with Error Injection
ch4/edactb_pkg.vhd	4.6.1-2	Support Package for EDAC Verificiation Modeling
ch4/Force_Pkg.vhd	4.6.1-3	Support Package for Error Injection Modeling
ch4/edacclient.vhd	4.6.1-4	EDAC Client Model
ch4/edacserver.vhd	4.6.1-5	EDAC Server Model
ch4/systemedac_tb.vhd	4.6.1-6	Top-Level Model
ch4/cdnedacvhd.do	4.6.1-7	EDAC testbench compilation script
ch4/edacmemtop.v	4.6.1-2	Testbench Memory for Testbench
ch4/edactask.v	4.6.2-1b	Job Structure
ch4/edacclient.v	4.6.2-3	Verilog Client for Testbench
ch4/edacserver.v	4.6.2-4	Verilog Server for Testbench
ch4/systemedac_tb.v	4.6.2-5	Top-Level Testbench
ch5/cpu_fsmfast.v	5.2.2.5-1	Verilog HDL of CPU FSM Design
ch5/cpu_fsmfasttb.v	5.2.2.6-1	Verilog Testbench
ch5/cpu_fsmfast.vhd	5.2.2.7-1	VHDL CPU Model
ch5/cpu_fsmfast_tb.vhd	5.2.2.8-1	VHDL Testbench
ch5/cpu_ucose.v	5.2.2.9-1	Verilog Architecture for Microcoded Architecture

ch5/rom_ucose.v	5.2.2.9-2	Sample Microcode Memory
ch5/cpuucodetop.v	5.2.2.9-3	Verilog Top Level Microcoded Architecture
ch6/Testminus.vhd	6.1.1-4	Subtractor with Operands of Unequal Sizes
ch6/ctestmultun.vhd	6.1.1-6	UNSIGNED Multiplication
Ch6/testgtrsg.vhd	6.1.1-9	SIGNED Comparison with Numbers of Unequal Lengths
ch6/testshlun.vhd	6.1.1-11	Shift Unsigned Example
ch6/testabs.vhd	6.1.1-13	ABSOLUTE ABS Example
ch6/arith2.v	6.2.1.1-1	Numbers on Registers and Integer Objects
ch6/why.v	6.2.1.1-2	Comparison Operations on Registers and Integer Objects
ch6/signmult9cc.v	6.2.1.2-1	Model of a Multiplier
ch6/signmult95_tb.v	6.2.1.2-2	Simple Multiplier Testbench
Ch6/test_signed1.v	6.2.2.1-1	Application of SIGNED Type
ch6/signed_counter.v	6.2.2.1-2	Counter and Testbench with SIGNED Ports and Registers
ch6/signed_shift.v	6.2.2.1-4	Verilog 2001 Shift Concepts
ch7/foo.v, ch7/bar.v	7.2-1	Sample Verilog Module
ch7/foo.vhd	7.2-2	File Generated by ncshell
ch7/bar.vhd	7.2-3	File Generated by ncshell
ch7/top.vhd	7.2-4	VHDL file instantiating the Verilog modules
ch7/foo2.vhd	7.3-1	Entity foo2.vhd
ch7/foo3.vhd	7.3.1-1	VHDL Model to be Instantiated into a Verilog Module
ch7/top3.v	7.3.1-2	Verilog Module Instantiating the VHDL Component
ch7/foo3.v	7.3.2-1	Ncshell Generated Verilog Model
ch9/testdelay.vhd	9.3.2-1	VHDL Update of Projected Waveforms
ch9/testdelay.v	9.3.2-2	Verilog Update of Projected Waveforms
ch9/Testblk.v	9.3.3-1	Simple always Block Example and Simulation Results
ch9/Testnblk.v	9.3.3-2	Simple always Non-Block Example and Simulation Results
ch9/testblock	9.3.3-3	Potential Errors when Blocking Assignments are

		Misunderstood
IEEE (Arithmetic Packages)		numeric_std.vhd numeric_signed.vhd numeric_unsigned.vhd numeric_bit.vhd numeric_extra.vhd std_logic_1164.vhd std_logic_arith.vhd std_logic_unsigned.vhd std_logic_misc.vhd std_logic_signed.vhd
DRAGONFLY		The DRAGONFLY micro core has been designed as a small (less than 4K gates), fast and programmable core, to be used in an ASIC or a FPGA, in areas such as serial communication management (UART, Smart cards controllers, LDC drivers, I <sup>2</sup> C controllers, SPI controllers), on-chip test and debug of complex blocks, intelligent DMA, FLASH controllers, audio-rate SDRAM controllers, and so on. VHDL Models and documentation.
DSP		DSP (DEMANDING SPACE-BASED PROCESSING!) THE PATH BEHIND AND THE ROAD AHEAD Dr SM Parkes, Applied Computing, University of Dundee, Dundee, DD1 4HN, Scotland, UK PDF Document
CliffCummings		<a href="http://www.sunburst-design.com">www.sunburst-design.com</a> Sunburst Design, Inc. Home Page <a href="http://www.sunburst-design.com/cliffc">www.sunburst-design.com/cliffc</a> Cliff's bio page <a href="http://www.sunburst-design.com/papers">www.sunburst-design.com/papers</a> Cliff's papers web page (to download papers, go to Cliff's papers web page and select one of the papers listed) - Coding And Scripting Techniques For FSM Designs With Synthesis-Optimized, - Glitch-Free Outputs Synthesis and Scripting Techniques for Designing Multi-Asynchronous Clock Designs - Nonblocking Assignments in Verilog Synthesis, Coding Styles That Kill! - Verilog-2001 Behavioral and Synthesis Enhancements RTL Coding Styles That Yield Simulation and Synthesis Mismatches Verilog Training
RamLib		Free IP, Free RAM Models and documentation (Ram and FIFO in VHDL)
risc8		Free IP, The Free-RISC8 is a Verilog synthesizable model of a simple 8-bit microcontroller. Ans_RISC8_Core – VHDL Model
6502		FREE IP, FREE-6502 in VHDL

<p><b>8051</b></p>		<p>VHDL 8051 processor</p>
<p><b>HDL_Info</b></p>	<p>Verilog_vs_vhdl                        VerilogHDL                      Coding_Motorola                      Verilog-Mode                      vlogfaq1,2,3                      vhdlfq1,2,3,4                      Vhdl-syntax_93                      Vhdl-syntax_93                      VHDL.hlp</p>	<p>VHDL &amp; Verilog Compared &amp; Contrasted Plus Modeled Example                      Written in VHDL, Verilog and C Douglas J. Smith, VeriBest                      Incorporated. Please fetch from  <a href="http://www.angelfire.com/in/rajesh52/verilogvhdl.html">http://www.angelfire.com/in/rajesh52/verilogvhdl.html</a>                      Verilog HDL Coding, Semiconductor Reuse Standard, Motorola                      Verilog-Mode for emacs (auto)                      Verilog FAQ                      VHDL FAQ                      VHDL Syntax                      VHDL Help</p>
<p><b>Synplicity</b></p>	<p>Launch.exe  </p>	<p><b>Get Synplify &amp; Synplify Pro</b>                      Synplicity University Program  <b>Literature</b>                      Synplify Datasheet                      Synplify Pro Datasheet                      Synplify/Amplify Family Brochure                      Success with MindTree Consulting                      Success with Sonus Networks                      Synplify web page                      Synplify Pro web page</p>
<p><b>Cadence</b></p>	<p>                      vlogrefTOC.html                      ncvlogTOC.html                      ncvlogPN                      ncvhdlTOC.html                      ncvhdlPN                      nccoexpNTOC.</p>	<p>See <a href="http://www.cadence.com/">http://www.cadence.com/</a>  <a href="http://www.cadence.com/products/index.html">http://www.cadence.com/products/index.html</a>                      Verilog-XL Reference, Product Version 3.3                      Cadence NC-Verilog Simulator Help, Product Version 3.3                      Cadence NC-Verilog Simulator Product Notes, Version 3.3                      Cadence NC-VHDL Simulator Help, Product Version 3.3                      Cadence NC-VHDL Simulator Product Notes, Version 3.3                      Cadence NC-Sim Mixed Language Simulator Notes, V 3.3                      For a guided tour of the Cadence VHDL and                      Verilog Desktop simulator  <a href="http://www.orcad.com/product/simulation/hdlsim/">http://www.orcad.com/product/simulation/hdlsim/</a></p>

<b>CPU-Design-HOWTO</b>	CPU-Design-HOWTO.html 	<i>The document has URL links to help students understand how a CPU is designed and manufactured.</i>
<b>CVS-RCS-HOWTO.html</b>	CVS-RCS-HOWTO.html	<i>This document is a "practical guide" to very quickly setup CVS/RCS source code control system</i>
<b>C++Programming-HOWTO</b>	C++Programming-HOWTO.html	<i>This document provides a comprehensive list of C++ URL pointers, links to C++ online textbooks, and programming tips on C++.</i>
<b>Emacs-Beginner-HOWTO</b>	Emacs-Beginner-HOWTO.html	<i>This document introduces users to the Emacs editor</i>
<b>emacs_tshell</b>		Emacs and Tshell + GNU tools for PC Windows 9X/NT/2000
<b>leon-2.2</b>		<i>From <a href="http://www.estec.esa.nl/wsmwww/leon/">http://www.estec.esa.nl/wsmwww/leon/</a> The LEON core is a <u>SPARC*</u> compatible integer unit developed for future space missions. It has been implemented as a highly configurable, synthesizable VHDL model. To promote the SPARC standard and enable development of system-on-a-chip (SOC) devices using SPARC cores, the <u>European Space Agency</u> is making the full source code freely available under the <u>GNU</u> LGPL license.</i>
<b>OpenMore</b>		OpenMore spreadsheet for grading of designs
<b>sutherland-hdl.html</b>		Link to Verilog 2001, Sutherland HDLCON paper Sutherland HDL, Inc Links to Verilog books and web sites, on-line Verilog reference, training, etc..

# Acknowledgements

*Real Chip Design and Verification Using Verilog and VHDL* could not have been written without the support of Synplicity, Cadence Design Systems, Cliff Cummings, Altera, Xilinx, and numerous papers and comments provided across the community and newsgroups.

I sincerely thank Synplicity<sup>5</sup> for extending a license of *Synplify Pro* FPGA synthesis tool. Synplicity is recognized as a vendor of advanced, efficient, and easy to use synthesis tools targeted for FPGAs, and now ASICs. *Synplify Pro* was an invaluable tool that enabled me to verify the synthesizability of models, to view and understand the RTL structures implied by the HDL, and to compare performance efficiency of potential architectures. In addition, I thank Andrew Dauman for reviewing the book and for all his support.

I genuinely thank Cadence Design Systems<sup>6</sup> for granting me a license of *NC-Sim* mixed mode HDL simulator and a license of *HAL* for HDL analysis and lint checking. Cadence Design Systems is a recognized leader in the EDA tool industry, and these licenses enabled me to verify the models and testbenches through simulation. I also thank Cadence for granting me permission to copy and provide on CD some of the reference documentation pertinent to specific topics. Martyn Pollard provided invaluable comments.

I gratefully thank Cliff Cummings<sup>7</sup> for granting permission to reprint some excellent papers on design methodologies, for his excellent review of my Verilog code, and for providing very useful suggestions on Verilog coding style and design alternatives.

I thank Altera and Xilinx for their permissions to incorporate in this book information contained in excellent white papers.

I thank Jeff Harris from Insight Design Services<sup>8</sup> for his permission to incorporate Rob Weinstein's *Flancter* design for the setting of flags in multiple clock domains.

I thank Richard Katz<sup>9</sup> for his contribution on the safe one-hot counter design. I thank users of *comp.lang.vhdl*, *comp.lang.Verilog*, *comp.arch.fpga* for sharing issues and solutions, many of which were included in this book.

I cannot leave out my son Michael Cohen for tapping into his though rough JAVA, C++, UML software architecture and design expertise.

I especially thank my wife, Gloria Jean, for supporting me in this endeavor.

---

<sup>5</sup> <http://www.synplicity.com/>

<sup>6</sup> <http://www.cadence.com/>

<sup>7</sup> <http://www.sunburst-design.com/> (*Expert Verilog, Synthesis and Verification Training*)

<sup>8</sup> <http://www.memecdesign.com/>

<sup>9</sup> <http://rk.gsfc.nasa.gov/>



**Sculpture Created by my Wife Gloria to  
Express my Long Hours with a Laptop in the Creation of HDL Books**

# About the Author

**Ben Cohen** is currently an HDL language trainer and consultant. He has technical experience in digital and analog hardware design, computer architecture, ASIC design, synthesis, and use of hardware description languages for modeling of statistical simulations, instruction set descriptions, and hardware models. He applied VHDL since 1990 to model various bus functional models of computer interfaces. He authored *VHDL Coding Styles and Methodologies*, first and second editions, and *VHDL Answers to Frequently Asked Questions*, first and second editions, and *Component Design by Example*. He was one of the pilot team members of the VHDL Synthesis Interoperability Working Group of the Design Automation Standards Committee who authored the *IEEE P1076.6 Standard for VHDL Register Transfer Level Synthesis*. He is currently a member of the *VHDL* and the *Verilog Synthesis Interoperability Working Group of the Design Automation Standards Committees*. He taught several VHDL training classes, and provided VHDL consulting services on several tasks.

**VhdlCohen Publishing**  
**Email:** [VhdlCohen@aol.com](mailto:VhdlCohen@aol.com)  
**Web page:** <http://www.vhdlcohen.com/>

# DISCLAIMER

Every attempt was made to ensure accuracy in the specifications and implementation of the models. However, All code provided in this book and in the accompanied CD is distributed with *\*ABSOLUTELY NO SUPPORT\** and *\*NO WARRANTY\** from the author. The author shall not be liable for damage in connection with, or arising out of, the furnishing, performance or use of the models provided in the book and CD.

The software media is distributed on an "AS IS" basis, without warranty.

If the media is defective, you may return it for a replacement.

Use or reproduction of the information provided in this book and on the enclosed CD for commercial gain is strictly prohibited.