
RTL HARDWARE DESIGN USING VHDL

**Coding for Efficiency, Portability,
and Scalability**

PONG P. CHU

Cleveland State University



A JOHN WILEY & SONS, INC., PUBLICATION

This Page Intentionally Left Blank

RTL HARDWARE DESIGN USING VHDL

This Page Intentionally Left Blank

RTL HARDWARE DESIGN USING VHDL

**Coding for Efficiency, Portability,
and Scalability**

PONG P. CHU

Cleveland State University



A JOHN WILEY & SONS, INC., PUBLICATION

Copyright © 2006 by John Wiley & Sons, Inc. All rights reserved.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.

Published simultaneously in Canada.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4470, or on the web at www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permission>.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services or for technical support, please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic format. For information about Wiley products, visit our web site at www.wiley.com.

Library of Congress Cataloging-in-Publication Data:

Chu, Pong P., 1959—

RTL hardware design using VHDL / by Pong P. Chu.

p. cm.

Includes bibliographical references and index.

“A Wiley-Interscience publication.”

ISBN-13: 978-0-471-72092-8 (alk. paper)

ISBN-10: 0-471-72092-5 (alk. paper)

1. Digital electronics—Data processing. 2. VHDL (Computer hardware description language). I. Title.

TK7868.D5C46 2006

621.39'2—dc22

2005054234

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

To my parents Chia-Chi and Chi-Te, my wife Lee, and my daughter Patricia

This Page Intentionally Left Blank

CONTENTS

Preface	xix
Acknowledgments	xxiii
1 Introduction to Digital System Design	1
1.1 Introduction	1
1.2 Device technologies	2
1.2.1 Fabrication of an IC	2
1.2.2 Classification of device technologies	2
1.2.3 Comparison of technologies	5
1.3 System representation	8
1.4 Levels of Abstraction	9
1.4.1 Transistor-level abstraction	10
1.4.2 Gate-level abstraction	10
1.4.3 Register-transfer-level (RT-level) abstraction	11
1.4.4 Processor-level abstraction	12
1.5 Development tasks and EDA software	12
1.5.1 Synthesis	13
1.5.2 Physical design	14
1.5.3 Verification	14
1.5.4 Testing	16
1.5.5 EDA software and its limitations	16
	vii

1.6	Development flow	17
1.6.1	Flow of a medium-sized design targeting FPGA	17
1.6.2	Flow of a large design targeting FPGA	19
1.6.3	Flow of a large design targeting ASIC	19
1.7	Overview of the book	20
1.7.1	Scope	20
1.7.2	Goal	20
1.8	Bibliographic notes	21
	Problems	22
2	Overview of Hardware Description Languages	23
2.1	Hardware description languages	23
2.1.1	Limitations of traditional programming languages	23
2.1.2	Use of an HDL program	24
2.1.3	Design of a modern HDL	25
2.1.4	VHDL	25
2.2	Basic VHDL concept via an example	26
2.2.1	General description	27
2.2.2	Structural description	30
2.2.3	Abstract behavioral description	33
2.2.4	Testbench	35
2.2.5	Configuration	37
2.3	VHDL in development flow	38
2.3.1	Scope of VHDL	38
2.3.2	Coding for synthesis	40
2.4	Bibliographic notes	40
	Problems	41
3	Basic Language Constructs of VHDL	43
3.1	Introduction	43
3.2	Skeleton of a basic VHDL program	44
3.2.1	Example of a VHDL program	44
3.2.2	Entity declaration	44
3.2.3	Architecture body	46
3.2.4	Design unit and library	46
3.2.5	Processing of VHDL code	47
3.3	Lexical elements and program format	47
3.3.1	Lexical elements	47
3.3.2	VHDL program format	49
3.4	Objects	51
3.5	Data types and operators	53

3.5.1	Predefined data types in VHDL	53
3.5.2	Data types in the IEEE std_logic_1164 package	56
3.5.3	Operators over an array data type	58
3.5.4	Data types in the IEEE numeric_std package	60
3.5.5	The std_logic_arith and related packages	64
3.6	Synthesis guidelines	65
3.6.1	Guidelines for general VHDL	65
3.6.2	Guidelines for VHDL formatting	66
3.7	Bibliographic notes	66
	Problems	66
4	Concurrent Signal Assignment Statements of VHDL	69
4.1	Combinational versus sequential circuits	69
4.2	Simple signal assignment statement	70
4.2.1	Syntax and examples	70
4.2.2	Conceptual implementation	70
4.2.3	Signal assignment statement with a closed feedback loop	71
4.3	Conditional signal assignment statement	72
4.3.1	Syntax and examples	72
4.3.2	Conceptual implementation	76
4.3.3	Detailed implementation examples	78
4.4	Selected signal assignment statement	85
4.4.1	Syntax and examples	85
4.4.2	Conceptual implementation	88
4.4.3	Detailed implementation examples	90
4.5	Conditional signal assignment statement versus selected signal assignment statement	93
4.5.1	Conversion between conditional signal assignment and selected signal assignment statements	93
4.5.2	Comparison between conditional signal assignment and selected signal assignment statements	94
4.6	Synthesis guidelines	95
4.7	Bibliographic notes	95
	Problems	95
5	Sequential Statements of VHDL	97
5.1	VHDL process	97
5.1.1	Introduction	97
5.1.2	Process with a sensitivity list	98
5.1.3	Process with a wait statement	99
5.2	Sequential signal assignment statement	100

5.3	Variable assignment statement	101
5.4	If statement	103
5.4.1	Syntax and examples	103
5.4.2	Comparison to a conditional signal assignment statement	105
5.4.3	Incomplete branch and incomplete signal assignment	107
5.4.4	Conceptual implementation	109
5.4.5	Cascading single-branched if statements	110
5.5	Case statement	112
5.5.1	Syntax and examples	112
5.5.2	Comparison to a selected signal assignment statement	114
5.5.3	Incomplete signal assignment	115
5.5.4	Conceptual implementation	116
5.6	Simple for loop statement	118
5.6.1	Syntax	118
5.6.2	Examples	118
5.6.3	Conceptual implementation	119
5.7	Synthesis of sequential statements	120
5.8	Synthesis guidelines	120
5.8.1	Guidelines for using sequential statements	120
5.8.2	Guidelines for combinational circuits	121
5.9	Bibliographic notes	121
	Problems	121
6	Synthesis Of VHDL Code	125
6.1	Fundamental limitations of EDA software	125
6.1.1	Computability	126
6.1.2	Computation complexity	126
6.1.3	Limitations of EDA software	128
6.2	Realization of VHDL operators	129
6.2.1	Realization of logical operators	129
6.2.2	Realization of relational operators	129
6.2.3	Realization of addition operators	130
6.2.4	Synthesis support for other operators	130
6.2.5	Realization of an operator with constant operands	130
6.2.6	An example implementation	131
6.3	Realization of VHDL data types	133
6.3.1	Use of the <code>std_logic</code> data type	133
6.3.2	Use and realization of the 'Z' value	133
6.3.3	Use of the '-' value	137
6.4	VHDL synthesis flow	139
6.4.1	RT-level synthesis	139
6.4.2	Module generator	141

6.4.3	Logic synthesis	142
6.4.4	Technology mapping	143
6.4.5	Effective use of synthesis software	148
6.5	Timing considerations	149
6.5.1	Propagation delay	150
6.5.2	Synthesis with timing constraints	154
6.5.3	Timing hazards	156
6.5.4	Delay-sensitive design and its dangers	158
6.6	Synthesis guidelines	160
6.7	Bibliographic notes	160
	Problems	160
7	Combinational Circuit Design: Practice	163
7.1	Derivation of efficient HDL description	163
7.2	Operator sharing	164
7.2.1	Sharing example 1	165
7.2.2	Sharing example 2	166
7.2.3	Sharing example 3	168
7.2.4	Sharing example 4	169
7.2.5	Summary	170
7.3	Functionality sharing	170
7.3.1	Addition–subtraction circuit	171
7.3.2	Signed–unsigned dual-mode comparator	173
7.3.3	Difference circuit	175
7.3.4	Full comparator	177
7.3.5	Three-function barrel shifter	178
7.4	Layout-related circuits	180
7.4.1	Reduced-xor circuit	181
7.4.2	Reduced-xor-vector circuit	183
7.4.3	Tree priority encoder	187
7.4.4	Barrel shifter revisited	192
7.5	General circuits	196
7.5.1	Gray code incrementor	196
7.5.2	Programmable priority encoder	199
7.5.3	Signed addition with status	201
7.5.4	Combinational adder-based multiplier	203
7.5.5	Hamming distance circuit	206
7.6	Synthesis guidelines	208
7.7	Bibliographic notes	208
	Problems	208
8	Sequential Circuit Design: Principle	213

8.1	Overview of sequential circuits	213
8.1.1	Sequential versus combinational circuits	213
8.1.2	Basic memory elements	214
8.1.3	Synchronous versus asynchronous circuits	216
8.2	Synchronous circuits	217
8.2.1	Basic model of a synchronous circuit	217
8.2.2	Synchronous circuits and design automation	218
8.2.3	Types of synchronous circuits	219
8.3	Danger of synthesis that uses primitive gates	219
8.4	Inference of basic memory elements	221
8.4.1	D latch	221
8.4.2	D FF	222
8.4.3	Register	225
8.4.4	RAM	225
8.5	Simple design examples	226
8.5.1	Other types of FFs	226
8.5.2	Shift register	229
8.5.3	Arbitrary-sequence counter	232
8.5.4	Binary counter	233
8.5.5	Decade counter	236
8.5.6	Programmable mod- m counter	237
8.6	Timing analysis of a synchronous sequential circuit	239
8.6.1	Synchronized versus unsynchronized input	239
8.6.2	Setup time violation and maximal clock rate	240
8.6.3	Hold time violation	243
8.6.4	Output-related timing considerations	243
8.6.5	Input-related timing considerations	244
8.7	Alternative one-segment coding style	245
8.7.1	Examples of one-segment code	245
8.7.2	Summary	250
8.8	Use of variables in sequential circuit description	250
8.9	Synthesis of sequential circuits	253
8.10	Synthesis guidelines	253
8.11	Bibliographic notes	253
	Problems	254
9	Sequential Circuit Design: Practice	257
9.1	Poor design practices and their remedies	257
9.1.1	Misuse of asynchronous signals	258
9.1.2	Misuse of gated clocks	260
9.1.3	Misuse of derived clocks	262
9.2	Counters	265

9.2.1	Gray counter	265
9.2.2	Ring counter	266
9.2.3	LFSR (linear feedback shift register)	269
9.2.4	Decimal counter	272
9.2.5	Pulse width modulation circuit	275
9.3	Registers as temporary storage	276
9.3.1	Register file	276
9.3.2	Register-based synchronous FIFO buffer	279
9.3.3	Register-based content addressable memory	287
9.4	Pipelined design	293
9.4.1	Delay versus throughput	294
9.4.2	Overview on pipelined design	294
9.4.3	Adding pipeline to a combinational circuit	297
9.4.4	Synthesis of pipelined circuits and retiming	307
9.5	Synthesis guidelines	308
9.6	Bibliographic notes	309
	Problems	309
10	Finite State Machine: Principle and Practice	313
10.1	Overview of FSMs	313
10.2	FSM representation	314
10.2.1	State diagram	315
10.2.2	ASM chart	317
10.3	Timing and performance of an FSM	321
10.3.1	Operation of a synchronous FSM	321
10.3.2	Performance of an FSM	324
10.3.3	Representative timing diagram	325
10.4	Moore machine versus Mealy machine	325
10.4.1	Edge detection circuit	326
10.4.2	Comparison of Moore output and Mealy output	328
10.5	VHDL description of an FSM	329
10.5.1	Multi-segment coding style	330
10.5.2	Two-segment coding style	333
10.5.3	Synchronous FSM initialization	335
10.5.4	One-segment coding style and its problem	336
10.5.5	Synthesis and optimization of FSM	337
10.6	State assignment	338
10.6.1	Overview of state assignment	338
10.6.2	State assignment in VHDL	339
10.6.3	Handling the unused states	341
10.7	Moore output buffering	342
10.7.1	Buffering by clever state assignment	342

10.7.2	Look-ahead output circuit for Moore output	344
10.8	FSM design examples	348
10.8.1	Edge detection circuit	348
10.8.2	Arbiter	353
10.8.3	DRAM strobe generation circuit	358
10.8.4	Manchester encoding circuit	363
10.8.5	FSM-based binary counter	367
10.9	Bibliographic notes	369
	Problems	369
11	Register Transfer Methodology: Principle	373
11.1	Introduction	373
11.1.1	Algorithm	373
11.1.2	Structural data flow implementation	374
11.1.3	Register transfer methodology	375
11.2	Overview of FSMD	376
11.2.1	Basic RT operation	376
11.2.2	Multiple RT operations and data path	378
11.2.3	FSM as the control path	379
11.2.4	ASMD chart	379
11.2.5	Basic FSMD block diagram	380
11.3	FSMD design of a repetitive-addition multiplier	382
11.3.1	Converting an algorithm to an ASMD chart	382
11.3.2	Construction of the FSMD	385
11.3.3	Multi-segment VHDL description of an FSMD	386
11.3.4	Use of a register value in a decision box	389
11.3.5	Four- and two-segment VHDL descriptions of FSMD	391
11.3.6	One-segment coding style and its deficiency	394
11.4	Alternative design of a repetitive-addition multiplier	396
11.4.1	Resource sharing via FSMD	396
11.4.2	Mealy-controlled RT operations	400
11.5	Timing and performance analysis of FSMD	404
11.5.1	Maximal clock rate	404
11.5.2	Performance analysis	407
11.6	Sequential add-and-shift multiplier	407
11.6.1	Initial design	408
11.6.2	Refined design	412
11.6.3	Comparison of three ASMD designs	417
11.7	Synthesis of FSMD	417
11.8	Synthesis guidelines	418
11.9	Bibliographic notes	418
	Problems	418

12 Register Transfer Methodology: Practice	421
12.1 Introduction	421
12.2 One-shot pulse generator	422
12.2.1 FSM implementation	422
12.2.2 Regular sequential circuit implementation	424
12.2.3 Implementation using RT methodology	425
12.2.4 Comparison	427
12.3 SRAM controller	430
12.3.1 Overview of SRAM	430
12.3.2 Block diagram of an SRAM controller	434
12.3.3 Control path of an SRAM controller	436
12.4 GCD circuit	445
12.5 UART receiver	455
12.6 Square-root approximation circuit	460
12.7 High-level synthesis	469
12.8 Bibliographic notes	470
Problems	470
 13 Hierarchical Design in VHDL	 473
13.1 Introduction	473
13.1.1 Benefits of hierarchical design	474
13.1.2 VHDL constructs for hierarchical design	474
13.2 Components	475
13.2.1 Component declaration	475
13.2.2 Component instantiation	477
13.2.3 Caveats in component instantiation	480
13.3 Generics	481
13.4 Configuration	485
13.4.1 Introduction	485
13.4.2 Configuration declaration	486
13.4.3 Configuration specification	488
13.4.4 Component instantiation and configuration in VHDL 93	488
13.5 Other supporting constructs for a large system	489
13.5.1 Library	489
13.5.2 Subprogram	491
13.5.3 Package	492
13.6 Partition	495
13.6.1 Physical partition	495
13.6.2 Logical partition	496
13.7 Synthesis guidelines	497
13.8 Bibliographic notes	497

Problems	497
14 Parameterized Design: Principle	499
14.1 Introduction	499
14.2 Types of parameters	500
14.2.1 Width parameters	500
14.2.2 Feature parameters	501
14.3 Specifying parameters	501
14.3.1 Generics	501
14.3.2 Array attribute	502
14.3.3 Unconstrained array	503
14.3.4 Comparison between a generic and an unconstrained array	506
14.4 Clever use of an array	506
14.4.1 Description without fixed-size references	507
14.4.2 Examples	509
14.5 For generate statement	512
14.5.1 Syntax	513
14.5.2 Examples	513
14.6 Conditional generate statement	517
14.6.1 Syntax	517
14.6.2 Examples	518
14.6.3 Comparisons with other feature-selection methods	525
14.7 For loop statement	528
14.7.1 Introduction	528
14.7.2 Examples of a simple for loop statement	528
14.7.3 Examples of a loop body with multiple signal assignment statements	530
14.7.4 Examples of a loop body with variables	533
14.7.5 Comparison of the for generate and for loop statements	536
14.8 Exit and next statements	537
14.8.1 Syntax of the exit statement	537
14.8.2 Examples of the exit statement	537
14.8.3 Conceptual implementation of the exit statement	539
14.8.4 Next statement	540
14.9 Synthesis of iterative structure	541
14.10 Synthesis guidelines	542
14.11 Bibliographic notes	542
Problems	542
15 Parameterized Design: Practice	545
15.1 Introduction	545

15.2	Data types for two-dimensional signals	546
15.2.1	Genuine two-dimensional data type	546
15.2.2	Array-of-arrays data type	548
15.2.3	Emulated two-dimensional array	550
15.2.4	Example	552
15.2.5	Summary	554
15.3	Commonly used intermediate-sized RT-level components	555
15.3.1	Reduced-xor circuit	555
15.3.2	Binary decoder	558
15.3.3	Multiplexer	560
15.3.4	Binary encoder	564
15.3.5	Barrel shifter	566
15.4	More sophisticated examples	569
15.4.1	Reduced-xor-vector circuit	570
15.4.2	Multiplier	572
15.4.3	Parameterized LFSR	586
15.4.4	Priority encoder	588
15.4.5	FIFO buffer	591
15.5	Synthesis of parameterized modules	599
15.6	Synthesis guidelines	599
15.7	Bibliographic notes	600
	Problems	600
16	Clock and Synchronization: Principle and Practice	603
16.1	Overview of a clock distribution network	603
16.1.1	Physical implementation of a clock distribution network	603
16.1.2	Clock skew and its impact on synchronous design	605
16.2	Timing analysis with clock skew	606
16.2.1	Effect on setup time and maximal clock rate	606
16.2.2	Effect on hold time constraint	609
16.3	Overview of a multiple-clock system	610
16.3.1	System with derived clock signals	611
16.3.2	GALS system	612
16.4	Metastability and synchronization failure	612
16.4.1	Nature of metastability	613
16.4.2	Analysis of $MTBF(T_r)$	614
16.4.3	Unique characteristics of $MTBF(T_r)$	616
16.5	Basic synchronizer	617
16.5.1	The danger of no synchronizer	617
16.5.2	One-FF synchronizer and its deficiency	617
16.5.3	Two-FF synchronizer	619
16.5.4	Three-FF synchronizer	620

16.5.5	Proper use of a synchronizer	621
16.6	Single enable signal crossing clock domains	623
16.6.1	Edge detection scheme	623
16.6.2	Level-alternation scheme	627
16.7	Handshaking protocol	630
16.7.1	Four-phase handshaking protocol	630
16.7.2	Two-phase handshaking protocol	637
16.8	Data transfer crossing clock domains	639
16.8.1	Four-phase handshaking protocol data transfer	641
16.8.2	Two-phase handshaking data transfer	650
16.8.3	One-phase data transfer	651
16.9	Data transfer via a memory buffer	652
16.9.1	FIFO buffer	652
16.9.2	Shared memory	660
16.10	Synthesis of a multiple-clock system	661
16.11	Synthesis guidelines	662
16.11.1	Guidelines for general use of a clock	662
16.11.2	Guidelines for a synchronizer	662
16.11.3	Guidelines for an interface between clock domains	662
16.12	Bibliographic notes	663
	Problems	663
References		665
Topic Index		667

PREFACE

With the maturity and availability of hardware description language (HDL) and synthesis software, using them to design custom digital hardware has become a mainstream practice. Because of the resemblance of an HDL code to a traditional program (such as a C program), some users believe incorrectly that designing hardware in HDL involves simply writing syntactically correct software code, and assume that the synthesis software can automatically derive the physical hardware. Unfortunately, synthesis software can only perform transformation and local optimization, and cannot convert a poor description into an efficient implementation. Without an understanding of the hardware architecture, the HDL code frequently leads to unnecessarily complex hardware, or may not even be synthesizable.

This book provides in-depth coverage on the systematical development and synthesis of efficient, portable and scalable register-transfer-level (RT-level) digital circuits using the VHDL hardware description language. RT-level design uses intermediate-sized components, such as adders, comparators, multiplexers and registers, to construct a digital system. It is the level that is most suitable and effective for today's synthesis software.

RT-level design and VHDL are two somewhat independent subjects. VHDL code is simply one of the methods to describe a hardware design. The same design can also be described by a schematic or code in other HDLs. VHDL and synthesis software will not lead automatically to a better or worse design. However, they can shield designers from low-level details and allow them to explore and research better architectures.

The emphasis of the book is on *hardware* rather than *language*. Instead of treating synthesis software as a mysterious black box and listing “recipe-like” codes, we explain the relationship between the VHDL constructs and the underlying hardware structure and illustrate how to explore the design space and develop codes that can be synthesized into efficient cell-level implementation. The discussion is independent of technology and can

be applied to both ASIC and FPGA devices. The VHDL codes listed in the book largely follow the IEEE 1076.6 RTL synthesis standard and can be accepted by most synthesis software. Most codes can be synthesized without modification by the free “demo-version” synthesis software provided by FPGA vendors.

Scope The book focuses primarily on the design and synthesis of RT-level circuits. A subset of VHDL is used to describe the design. The book is not intended to be a comprehensive ASIC or FPGA book. All other issues, such as device architecture, placement and routing, simulation and testing, are discussed exclusively from the context of RT-level design.

Unique features The book is a hardware design text. VHDL and synthesis software are used as tools to realize the intended design. Several unique features distinguish the book:

- Suggest a coding style that shows a clear relationship between VHDL constructs and hardware components.
- Use easy-to-understand conceptual diagrams, rather than cell-level netlists, to explain the realization of VHDL codes.
- Emphasize the reuse aspect of the codes throughout the book.
- Consider RT-level design as an integral part of the overall development process and introduce good design practices and guidelines to ensure that an RT-level description can accommodate future simulation, verification and testing needs.
- Make the design “technology neutral” so that the developed VHDL code can be applied to both ASIC and FPGA devices.
- Follow the IEEE 1076.6 RTL synthesis standard to make the codes independent of synthesis software.
- Provide a set of synthesis guidelines at the end of each chapter.
- Contain a large number of non-trivial, practical examples to illustrate and reinforce the design concepts, procedures and techniques.
- Include two chapters on realizing sequential algorithms in hardware (known as “register transfer methodology”) and on designing control path and data path.
- Include two chapters on the scalable and parameterized designs and coding.
- Include a chapter on the synchronization and interface between multiple clock domains.

Book organization The book is basically divided into three major parts. The first part, Chapters 1 to 6, provides a comprehensive overview of VHDL and the synthesis process, and examines the hardware implementation of basic VHDL language constructs. The second part, Chapters 7 to 12, covers the core of the RT-level design, including combinational circuits, “regular” sequential circuits, finite state machine and circuits designed by register transfer methodology. The third part, Chapters 13 to 16, covers the system issues, including the hierarchy, parameterized and scalable design, and interface between clock domains. More detailed descriptions of the chapters follow.

- Chapter 1 presents a “big picture” of digital system design, including an overview on device technologies, system representation, development flow and software tools.
- Chapter 2 provides an overview on the design, usage and capability of a hardware description language. A series of simple codes is used to introduce the basic modeling concepts of VHDL.
- Chapter 3 provides an overview of the basic language constructs of VHDL, including lexical elements, objects, data types and operators. Because VHDL is a strongly typed language, the data types and operators are discussed in more detail.

- Chapter 4 covers the syntax, usage and implementation of concurrent signal assignment statements of VHDL. It shows how to realize these constructs by multiplexing and priority routing networks.
- Chapter 5 examines the syntax, usage and implementation of sequential statements of VHDL. It shows the realization of the sequential statements and discusses the caveats of using these statements.
- Chapter 6 explains the realization of VHDL operators and data types, provides an in-depth overview on the synthesis process and discusses the timing issue involved in synthesis.
- Chapter 7 covers the construction and VHDL description of more sophisticated combinational circuits. Examples show how to transform conceptual ideas into hardware, and illustrate resource-sharing and circuit-shaping techniques to reduce circuit size and increase performance.
- Chapter 8 introduces the synchronous design methodology and the construction and coding of synchronous sequential circuits. Basic “regular” sequential circuits, such as counters and shift registers, in which state transitions exhibit a regular pattern, are examined.
- Chapter 9 explores more sophisticated regular sequential circuits. The design examples show the implementation of a variety of counters, the use of registers as fast, temporary storage, and the construction of pipelined combinational circuits.
- Chapter 10 covers finite state machine (FSM), which is a sequential circuit with “random” transition patterns. The representation, timing and implementation issues of FSMs are studied with an emphasis on its use as the control circuit for a large, complex system.
- Chapter 11 introduces the register transfer methodology, which describes system operation by a sequence of data transfers and manipulations among registers, and demonstrates the construction of the data path (a regular sequential circuit) and the control path (an FSM) used in this methodology.
- Chapter 12 uses a variety of design examples to illustrate how the register transfer methodology can be used in various types of problems and to highlight the design procedure and relevant issues.
- Chapter 13 features the design hierarchy, in which a system is gradually divided into smaller parts. Mechanisms and language constructs of VHDL used to specify and configure a hierarchy are examined.
- Chapter 14 introduces parameterized design, in which the width and functionality of a circuit are specified by explicit parameters. Simple examples illustrate the mechanisms used to pass and infer parameters and the language constructs used to describe the replicated structures.
- Chapter 15 provides more sophisticated parameterized design examples. The main focus is on the derivation of efficient parameterized RT-level modules that can be used as building blocks of larger systems.
- Chapter 16 covers the effect of a non-ideal clock signal and discusses the synchronization of an asynchronous signal and the interface between two independent clock domains.

Audience The intended audience for the book is students in advanced digital system design course and practicing engineers who wish to sharpen their design skills or to learn the effective use of today’s synthesis software. Readers need to have basic knowledge of digital systems. The material is normally covered in an introductory digital design course,

which is a standard part in all electrical engineering and computer engineering curricula. No prior experience on HDL or synthesis is needed.

Verilog is another popular HDL. Since the book emphasizes hardware and methodology rather than language constructs, readers with prior Verilog experience can easily follow the discussion and learn VHDL along the way. Most VHDL codes can easily be translated into the Verilog language.

Web site An accompanying web site (http://academic.csuohio.edu/chu_p/rtl) provides additional information, including the following materials:

- Errata.
- Summary of coding guidelines.
- Code listing.
- Links to demo-version synthesis software.
- Links to some referenced materials.
- Frequently asked questions (FAQ) on RTL synthesis.
- Lecture slides for instructors.

Errata The book is “self-prepared,” which means the author has prepared all materials, including the illustrations, tables, code listing, indexing and formatting, by himself. As the errors are always bound to happen, the accompanying web site provides an updated errata sheet and a place to report errors.

P. P. CHU

Cleveland, Ohio
January 2006

ACKNOWLEDGMENTS

The author would like to express his gratitude to Professor George L. Kramerich for his encouragement and help during the course of this project. The work was partially supported by educational material development grant 0126752 from the National Science Foundation and a Teaching Enhancement grant from Cleveland State University.

P. P. Chu

This Page Intentionally Left Blank

CHAPTER 1

INTRODUCTION TO DIGITAL SYSTEM DESIGN

Developing and producing a digital system is a complicated process and involves many tasks. The design and synthesis of a register transfer level circuit, which is the focus of this book, is only one of the tasks. In this chapter, we present an overview of device technologies, system representation, development flow and software tools. This helps us to better understand the role of the design and synthesis task in the overall development and production process.

1.1 INTRODUCTION

Digital hardware has experienced drastic expansion and improvement in the past 40 years. Since its introduction, the number of transistors in a single chip has grown exponentially, and a silicon chip now routinely contains hundreds of thousands or even hundreds of millions of transistors. In the past, the major applications of digital hardware were computational systems. However, as the chip became smaller, faster, cheaper and more capable, many electronic, control, communication and even mechanical systems have been “digitized” internally, using digital circuits to store, process and transmit information.

As applications become larger and more complex, the task of designing digital circuits becomes more difficult. The best way to handle the complexity is to view the circuit at a more abstract level and utilize software tools to derive the low-level implementation. This approach shields us from the tedious details and allows us to concentrate and explore high-level design alternatives. Although software tools can automate certain tasks, they are capable of performing only limited transformation and optimization. They cannot, and

will not, do the design or convert a poor design to a good one. The ultimate efficiency still comes from human ingenuity and experience. The goal of this book is to show how to systematically develop an efficient, portable design description that is both abstract, yet detailed enough for effective software synthesis.

Developing and producing a digital circuit is a complicated process, and the design and synthesis are only two of the tasks. We should be aware of the “big picture” so that the design and synthesis can be efficiently integrated into the overall development and production process. The following sections provide an overview of device technologies, system representation, abstraction, development flow, and the use and limitations of software tools.

1.2 DEVICE TECHNOLOGIES

If we want to build a custom digital system, there are varieties of device technologies to choose, from off-the-shelf simple field-programmable components to full-custom devices that tailor the application down to the transistor level. There is no single best technology, and we have to consider the trade-offs among various factors, including chip area, speed, power and cost.

1.2.1 Fabrication of an IC

To better understand the differences between the device technologies, it is helpful to have a basic idea of the fabrication process of an integrated circuit (IC). An IC is made from layers of doped silicon, polysilicon, metal and silicon dioxide, built on top of one another, on a thin silicon wafer. Some of these layers form transistors, and others form planes of connection wires.

The basic step in IC fabrication is to construct a layer with a customized pattern, a process known as *lithography*. The pattern is defined by a *mask*. Today’s IC device technology typically consists of 10 to 15 layers, and thus the lithography process has to be repeated 10 to 15 times during the fabrication of an IC, each time with a unique mask.

One important aspect of a device technology is the silicon area used by a circuit. It is expressed by the length of a smallest transistor that can be fabricated, usually measured in *microns* (a millionth of a meter). As the device fabrication process improved, the transistor size continued to shrink and now approaches a tenth of a micron.

1.2.2 Classification of device technologies

There is an array of device technologies that can be used to construct a custom digital circuit. One major characteristic of a technology is how the customization is done. In certain technologies, all the layers of a device are predetermined, and thus the device can be prefabricated and manufactured as a standard off-the-shelf part. The customization of a circuit can be performed “in the field,” normally by downloading a connection pattern to the device’s internal memory or by “burning the internal silicon fuses.” On the other hand, some device technologies need one or more layers to be customized for a particular application. The customization involves the creation of tailored masks and fabrication of the patterned layers. This process is expensive and complex and can only be done in a fabrication plant (known as a *foundry* or a *fab*). Thus, whether a device needed to be fabricated in a fab is the most important characteristic of a technology. In this book, we use

the term *application-specific IC (ASIC)* to represent device technologies that require a fab to do customization.

With an understanding of the difference between ASIC and non-ASIC, we can divide the device technologies further into the following types:

- Full-custom ASIC
- Standard-cell ASIC
- Gate array ASIC
- Complex field-programmable logic device
- Simple field-programmable logic device
- Off-the-shelf small- and medium-scaled IC (SSI/MSI) components

Full-custom ASIC In *full-custom ASIC* technology, all aspects of a digital circuit are tailored for one particular application. We have complete control of the circuit and can even craft the layout of a transistor to meet special area or performance needs. The resulting circuit is fully optimized and has the best possible performance. Unfortunately, designing a circuit at the transistor level is extremely complex and involved, and is only feasible for a small circuit. It is not practical to use this approach to design a complete system, which now may contain tens and even hundreds of millions of transistors. The major application of full-custom ASIC technology is to design the basic logic components that can be used as building blocks of a larger system. Another application is to design special-purpose “bit-slice” typed circuits, such as a 1-bit memory or 1-bit adder. These circuits have a regular structure and are constructed through a cascade of identical slices. To obtain optimal performance, full-custom ASIC technology is frequently used to design a single slice. The slice is then replicated a number of times to form a complete circuit.

The layouts of a full-custom ASIC chip are tailored to a particular application. All layers are different and a mask is required for every layer. During fabrication, all layers have to be custom constructed, and nothing can be done in advance.

Standard-cell ASIC In *standard-cell ASIC* (also simply known as *standard-cell*) technology, a circuit is constructed by using a set of predefined logic components, known as *standard cells*. These cells are predesigned and their layouts are validated and tested. Standard-cell ASIC technology allows us to work at the gate level rather than at the transistor level and thus greatly simplifies the design process. The device manufacturer usually provides a library of standard cells as the basic building blocks. The library normally consists of basic logic gates, simple combinational components, such as an and-or-inverter, 2-to-1 multiplexer and 1-bit full adder, and basic memory elements, such as a D-type latch and D-type flip-flop. Some libraries may also contain more sophisticated function blocks, such as an adder, barrel shifter and random access memory (RAM).

In standard-cell technology, a circuit is made of cells. The types of cells and the interconnection depend on the individual application. Whereas the layout of a cell is predetermined, the layout of the complete circuit is unique for a particular application and nothing can be constructed in advance. Thus, fabrication of a standard-cell chip is identical to that of a full-custom ASIC chip, and all layers have to be custom constructed.

Gate array ASIC In *gate array ASIC* (also simply known as *gate array*) technology, a circuit is built from an array of predefined cells. Unlike standard-cell technology, a gate array chip consists of only one type of cell, known as a *base cell*. The base cell is fairly simple, resembling a logic gate. Base cells are prearranged and placed in fixed positions, aligned as a one- or two-dimensional array. Since the location and type are predetermined,

the base cells can be prefabricated. The customization of a circuit is done by specifying the interconnect between these cells. A gate array vendor also provides a library of predesigned components, known as *macro cells*, which are built from base cells. The macro cells have a predefined interconnect and provide the designer with more sophisticated logic blocks.

Compared to standard-cell technology, the fabrication of a gate array device is much simpler, due to its fixed array structure. Since the array is common to all applications, the cell (and transistors) can be fabricated in advance. During construction of a chip, only the masks of metal layers, which specify the interconnect, are unique for an application and therefore must be customized. This reduces the number of custom layers from 10 to 15 layers to 3 to 5 layers and simplifies the fabrication process significantly.

Complex field-programmable device We now examine several non-ASIC technologies. The most versatile non-ASIC technology is the complex field-programmable device. In this technology, a device consists of an array of generic logic cells and general interconnect structure. Although the logic cells and interconnect structure are prefabricated, both are *programmable*. The programmability is obtained by utilizing semiconductor “fuses” or “switches,” which can be set as open- or short-circuit. The customization is done by configuring the device with a specific fuse pattern. This process can be accomplished by a simple, inexpensive device programmer, normally constructed as an add-on card or an adaptor cable of a PC. Since the customization is done “in the field” rather than “in a fab,” this technology is known as *field programmable*. (In contrast, ASIC technologies are “programmed” via one or more tailored masks and thus are *mask programmable*.)

The basic structures of gate array ASICs and complex field-programmable devices are somewhat similar. However, the interconnect structure of field-programmable devices is predetermined and thus imposes more constraints on signal routing. To reduce the amount of connection, more functionality is built into the logic cells of a field-programmable device, making a logic cell much more complex than a base cell or a standard cell of ASIC. According to the complexity and structure of logic cells, complex field-programmable devices can be divided roughly into two broad categories: *complex programmable logic device (CPLD)* and *field programmable gate array (FPGA)*.

The logic cell of a CPLD device is more sophisticated, normally consisting of a D-type flip-flop and a PAL-like unit with configurable product terms. The interconnect structure of a CPLD device tends to be more centralized, with few groups of concentrated routing lines. On the other hand, the logic cell of an FPGA device is usually smaller, typically including a D-type flip-flop and a small look-up table or a set of multiplexers. The interconnect structure between the cells tends to be distributed and more flexible. Because of its distributive nature, FPGA is better suited for large, high-capacity complex field-programmable devices.

Simple field-programmable device Simple field-programmable logic devices, as the name indicates, are programmable devices with simpler internal structure. Historically, these devices are generically called *programmable logic devices (PLDs)*. We add the word *simple* to distinguish them from FPGA and CPLD devices. Simple field-programmable devices are normally constructed as a two-level array, with an *and plane* and an *or plane*. The interconnect of one or both planes can be programmed to perform a logic function expressed in sum-of-product format. The devices include *programmable read only memory (PROM)*, in which the or plane can be programmed; *programmable array logic (PAL)*, in which the and plane can be programmed; and *programmable logic array (PLA)*, in which both planes can be programmed.

Unlike FPGA and CPLD devices, simple field-programmable logic devices do not have a general interconnect structure, and thus their functionality is severely limited. They are