
INTRODUCTION TO LOGIC CIRCUITS & LOGIC DESIGN WITH VERILOG

INTRODUCTION TO LOGIC CIRCUITS & LOGIC DESIGN WITH VERILOG

2ND EDITION

Brock J. LaMeres

 Springer

Brock J. LaMeres
Department of Electrical & Computer Engineering
Montana State University
Bozeman, MT, USA

ISBN 978-3-030-13604-8 ISBN 978-3-030-13605-5 (eBook)
<https://doi.org/10.1007/978-3-030-13605-5>

Library of Congress Control Number: 2019934718

© Springer Nature Switzerland AG 2019

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

The overall goal of this book is to fill a void that has appeared in the instruction of digital circuits over the past decade due to the rapid abstraction of system design. Up until the mid-1980s, digital circuits were designed using *classical* techniques. Classical techniques relied heavily on manual design practices for the synthesis, minimization, and interfacing of digital systems. Corresponding to this design style, academic textbooks were developed that taught classical digital design techniques. Around 1990, large-scale digital systems began being designed using hardware description languages (HDL) and automated synthesis tools. Broad-scale adoption of this *modern design* approach spread through the industry during this decade. Around 2000, hardware description languages and the modern digital design approach began to be taught in universities, mainly at the senior and graduate level. There were a variety of reasons that the modern digital design approach did not penetrate the lower levels of academia during this time. First, the design and simulation tools were difficult to use and overwhelmed freshman and sophomore students. Second, the ability to implement the designs in a laboratory setting was infeasible. The modern design tools at the time were targeted at custom integrated circuits, which are cost- and time-prohibitive to implement in a university setting. Between 2000 and 2005, rapid advances in programmable logic and design tools allowed the modern digital design approach to be implemented in a university setting, even in lower-level courses. This allowed students to learn the modern design approach based on HDLs and prototype their designs in real hardware, mainly field-programmable gate arrays (FPGAs). This spurred an abundance of textbooks to be authored, teaching hardware description languages and higher levels of design abstraction. This trend has continued until today. While abstraction is a critical tool for engineering design, the rapid movement toward teaching only the modern digital design techniques has left a void for freshman- and sophomore-level courses in digital circuitry. Legacy textbooks that teach the classical design approach are outdated and do not contain sufficient coverage of HDLs to prepare the students for follow-on classes. Newer textbooks that teach the modern digital design approach move immediately into high-level behavioral modeling with minimal or no coverage of the underlying hardware used to implement the systems. As a result, students are not being provided the resources to understand the fundamental hardware theory that lies beneath the modern abstraction such as interfacing, gate-level implementation, and technology optimization. Students moving too rapidly into high levels of abstraction have little understanding of what is going on when they click the “compile and synthesize” button of their design tool. This leads to graduates who can model a breadth of different systems in an HDL but have no depth into how the system is implemented in hardware. This becomes problematic when an issue arises in a real design and there is no foundational knowledge for the students to fall back on in order to debug the problem.

This new book addresses the lower-level foundational void by providing a comprehensive, bottom-up, coverage of digital systems. The book begins with a description of lower-level hardware including binary representations, gate-level implementation, interfacing, and simple combinational logic design. Only after a foundation has been laid in the underlying hardware theory is the Verilog language introduced. The Verilog introduction gives only the basic concepts of the language in order to model, simulate, and synthesize combinational logic. This allows the students to gain familiarity with the language and the modern design approach without getting overwhelmed by the full capability of the language. The book then covers sequential logic and finite-state machines at the structural level. Once this secondary foundation has been laid, the remaining capabilities of Verilog are presented that allow sophisticated, synchronous systems to be modeled. An entire chapter is then dedicated to examples of sequential system modeling, which allows the students to learn by example. The second part of the textbook introduces the details of programmable logic, semiconductor memory, and arithmetic circuits. The book culminates with a discussion of computer system design, which incorporates all of the

knowledge gained in the previous chapters. Each component of a computer system is described with an accompanying Verilog implementation, all while continually reinforcing the underlying hardware beneath the HDL abstraction.

Written the Way It Is Taught

The organization of this book is designed to follow the way in which the material is actually learned. Topics are presented only once sufficient background has been provided by earlier chapters to fully understand the material. An example of this *learning-oriented* organization is how the Verilog language is broken into two chapters. Chapter 5 presents an introduction to Verilog and the basic constructs to model combinational logic. This is an ideal location to introduce the language because the reader has just learned about combinational logic theory in Chap. 4. This allows the student to begin gaining experience using the Verilog simulation tools on basic combinational logic circuits. The more advanced constructs of Verilog, such as sequential modeling and test benches, are presented in Chap. 8 only after a thorough background in sequential logic is presented in Chap. 7. Another example of this learning-oriented approach is how arithmetic circuits are not introduced until Chap. 12. While technically the arithmetic circuits in Chap. 12 are combinational logic circuits and could be presented in Chap. 4, the student does not have the necessary background in Chap. 4 to fully understand the operation of the arithmetic circuitry, so its introduction is postponed.

This incremental, *just-in-time* presentation of material allows the book to follow the way the material is actually taught in the classroom. This design also avoids the need for the instructor to assign sections that move back and forth through the text. This not only reduces course design effort for the instructor but allows the student to know where they are in the sequence of learning. At any point, the student should know the material in prior chapters and be moving toward understanding the material in subsequent ones.

An additional advantage of this book's organization is that it supports giving the student hands-on experience with digital circuitry for courses with an accompanying laboratory component. The flow is designed to support lab exercises that begin using discrete logic gates on a breadboard and then move into HDL-based designs implemented on off-the-shelf FPGA boards. Using this approach to a laboratory experience gives the student experience with the basic electrical operation of digital circuits, interfacing, and HDL-based designs.

Learning Outcomes

Each chapter begins with an explanation of its learning objective followed by a brief preview of the chapter topics. The specific learning outcomes are then presented for the chapter in the form of concise statements about the measurable knowledge and/or skills the student will be able to demonstrate by the end of the chapter. Each section addresses a single, specific learning outcome. This eases the process of assessment and gives specific details on student performance. There are over 1000 assessment tools in the form of exercise problems and concept check questions that are tied directly to specific learning outcomes for both formative and summative assessment.

Teaching by Example

With nearly 250 worked examples, concept checks for each section, 200+ supporting figures, and 1000+ assessment problems, students are provided with multiple ways to learn. Each topic is described in a clear, concise written form with accompanying figures as necessary. This is then followed by annotated worked examples that match the form of the exercise problems at the end of each chapter. Additionally, concept check questions are placed at the end of each section in the book to measure the

student's general understanding of the material using a concept inventory assessment style. These features provide the student multiple ways to learn the material and build an understanding of digital circuitry.

Course Design

The book can be used in multiple ways. The first is to use the book to cover two, semester-based college courses in digital logic. The first course in this sequence is an *introduction to logic circuits* and covers Chaps. 1, 2, 3, 4, 5, 6, and 7. This introductory course, which is found in nearly all accredited electrical and computer engineering programs, gives students a basic foundation in digital hardware and interfacing. Chapters 1, 2, 3, 4, 5, 6, and 7 only cover relevant topics in digital circuits to make room for a thorough introduction to Verilog. At the end of this course, students have a solid foundation in digital circuits and are able to design and simulate Verilog models of concurrent and hierarchical systems. The second course in this sequence covers *logic design* using Chaps. 8, 9, 10, 11, 12, and 13. In this second course, students learn the advanced features of Verilog such as procedural assignments, sequential behavioral modeling, system tasks, and test benches. This provides the basis for building larger digital systems such as registers, finite-state machines, and arithmetic circuits. Chapter 13 brings all of the concepts together through the design of a simple 8-bit computer system that can be simulated and implemented using many off-the-shelf FPGA boards.

This book can also be used in a more accelerated digital logic course that reaches a higher level of abstraction in a single semester. This is accomplished by skipping some chapters and moving quickly through others. In this use model, it is likely that Chap. 2 on number systems and Chap. 3 on digital circuits would be quickly referenced but not covered in detail. Chapters 4 and 7 could also be covered quickly in order to move rapidly into Verilog modeling without spending significant time looking at the underlying hardware implementation. This approach allows a higher level of abstraction to be taught but provides the student with the reference material so that they can delve into the details of the hardware implementation if interested.

All exercise and concept problems that do not involve a Verilog model are designed so that they can be implemented as a multiple-choice or numeric entry question in a standard course management system. This allows the questions to be automatically graded. For the Verilog design questions, it is expected that the students will upload their Verilog source files and screenshots of their simulation waveforms to the course management system for manual grading by the instructor or teaching assistant.

Instructor Resources

Instructors adopting this book can access a growing collection of supplementary learning resources including *YouTube* videos created by the author, a solutions manual, a laboratory manual, and Verilog test benches for all problems. Additional resources are made available as demand grows. The growing library of YouTube videos can provide supplementary learning materials for students or facilitate fully online or flipped delivery of this material. The videos are found at https://www.youtube.com/c/DigitalLogicProgramming_LaMeres. The solutions manual contains a graphic-rich description of select exercise problems. A complementary lab manual has also been developed to provide additional learning activities based on both the 74HC discrete logic family and an off-the-shelf FPGA board. This manual is provided separately from the book in order to support the ever-changing technology options available for laboratory exercises.

What's New in the Second Edition

The most common request from adopters of the first edition of this book was more assessment problems and accompanying videos. As a result, the second edition now contains over 1000 assessment questions and a growing library of YouTube videos. Additionally, more worked examples have been added so that every section has abundant examples of how to apply the content to designing and analyzing digital circuits.

Bozeman, MT, USA

Brock J. LaMer

Acknowledgment

For JoAnn, Alexis, and Kylie. Thank you for your endless support of this project. You are my inspiration.

Contents

1: INTRODUCTION: ANALOG VERSUS DIGITAL	1
1.1 DIFFERENCES BETWEEN ANALOG AND DIGITAL SYSTEMS	1
1.2 ADVANTAGES OF DIGITAL SYSTEMS OVER ANALOG SYSTEMS	3
2: NUMBER SYSTEMS	7
2.1 POSITIONAL NUMBER SYSTEMS	7
2.1.1 <i>Generic Structure</i>	8
2.1.2 <i>Decimal Number System (Base 10)</i>	9
2.1.3 <i>Binary Number System (Base 2)</i>	9
2.1.4 <i>Octal Number System (Base 8)</i>	10
2.1.5 <i>Hexadecimal Number System (Base 16)</i>	10
2.2 BASE CONVERSION	11
2.2.1 <i>Converting to Decimal</i>	11
2.2.2 <i>Converting from Decimal</i>	14
2.2.3 <i>Converting Between 2^n Bases</i>	18
2.3 BINARY ARITHMETIC	22
2.3.1 <i>Addition (Carries)</i>	22
2.3.2 <i>Subtraction (Borrows)</i>	23
2.4 UNSIGNED AND SIGNED NUMBERS	25
2.4.1 <i>Unsigned Numbers</i>	25
2.4.2 <i>Signed Numbers</i>	26
3: DIGITAL CIRCUITRY AND INTERFACING	43
3.1 BASIC GATES	43
3.1.1 <i>Describing the Operation of a Logic Circuit</i>	43
3.1.2 <i>The Buffer</i>	45
3.1.3 <i>The Inverter</i>	46
3.1.4 <i>The AND Gate</i>	46
3.1.5 <i>The NAND Gate</i>	47
3.1.6 <i>The OR Gate</i>	47
3.1.7 <i>The NOR Gate</i>	47
3.1.8 <i>The XOR Gate</i>	48
3.1.9 <i>The XNOR Gate</i>	49
3.2 DIGITAL CIRCUIT OPERATION	50
3.2.1 <i>Logic Levels</i>	51
3.2.2 <i>Output DC Specifications</i>	51
3.2.3 <i>Input DC Specifications</i>	53
3.2.4 <i>Noise Margins</i>	53
3.2.5 <i>Power Supplies</i>	54
3.2.6 <i>Switching Characteristics</i>	56
3.2.7 <i>Data Sheets</i>	57

3.3	LOGIC FAMILIES	62
3.3.1	<i>Complementary Metal-Oxide Semiconductors (CMOS)</i>	62
3.3.2	<i>Transistor-Transistor Logic (TTL)</i>	71
3.3.3	<i>The 7400 Series Logic Families</i>	73
3.4	DRIVING LOADS	77
3.4.1	<i>Driving Other Gates</i>	77
3.4.2	<i>Driving Resistive Loads</i>	79
3.4.3	<i>Driving LEDs</i>	81
4:	COMBINATIONAL LOGIC DESIGN	93
4.1	BOOLEAN ALGEBRA	93
4.1.1	<i>Operations</i>	94
4.1.2	<i>Axioms</i>	94
4.1.3	<i>Theorems</i>	95
4.1.4	<i>Functionally Complete Operation Sets</i>	110
4.2	COMBINATIONAL LOGIC ANALYSIS	111
4.2.1	<i>Finding the Logic Expression from a Logic Diagram</i>	111
4.2.2	<i>Finding the Truth Table from a Logic Diagram</i>	112
4.2.3	<i>Timing Analysis of a Combinational Logic Circuit</i>	113
4.3	COMBINATIONAL LOGIC SYNTHESIS	115
4.3.1	<i>Canonical Sum of Products</i>	115
4.3.2	<i>The Minterm List (Σ)</i>	116
4.3.3	<i>Canonical Product of Sums (POS)</i>	118
4.3.4	<i>The Maxterm List (Π)</i>	120
4.3.5	<i>Minterm and Maxterm List Equivalence</i>	122
4.4	LOGIC MINIMIZATION	124
4.4.1	<i>Algebraic Minimization</i>	124
4.4.2	<i>Minimization Using Karnaugh Maps</i>	125
4.4.3	<i>Don't Cares</i>	137
4.4.4	<i>Using XOR Gates</i>	138
4.5	TIMING HAZARDS AND GLITCHES	141
5:	VERILOG (PART 1)	153
5.1	HISTORY OF HARDWARE DESCRIPTION LANGUAGES	154
5.2	HDL ABSTRACTION	157
5.3	THE MODERN DIGITAL DESIGN FLOW	161
5.4	VERILOG CONSTRUCTS	164
5.4.1	<i>Data Types</i>	165
5.4.2	<i>The Module</i>	168
5.4.3	<i>Verilog Operators</i>	171
5.5	MODELING CONCURRENT FUNCTIONALITY IN VERILOG	175
5.5.1	<i>Continuous Assignment</i>	176
5.5.2	<i>Continuous Assignment with Logical Operators</i>	176
5.5.3	<i>Continuous Assignment with Conditional Operators</i>	177
5.5.4	<i>Continuous Assignment with Delay</i>	179

5.6	STRUCTURAL DESIGN AND HIERARCHY	182
5.6.1	<i>Lower-Level Module Instantiation</i>	182
5.6.2	<i>Gate-Level Primitives</i>	184
5.6.3	<i>User-Defined Primitives</i>	185
5.6.4	<i>Adding Delay to Primitives</i>	186
5.7	OVERVIEW OF SIMULATION TEST BENCHES	187
6:	MSI LOGIC	195
6.1	DECODERS	195
6.1.1	<i>Example: One-Hot Decoder</i>	195
6.1.2	<i>Example: 7-Segment Display Decoder</i>	198
6.2	ENCODERS	202
6.2.1	<i>Example: One-Hot Binary Encoder</i>	203
6.3	MULTIPLEXERS	204
6.4	DEMULTIPLEXERS	207
7:	SEQUENTIAL LOGIC DESIGN	213
7.1	SEQUENTIAL LOGIC STORAGE DEVICES	213
7.1.1	<i>The Cross-Coupled Inverter Pair</i>	213
7.1.2	<i>Metastability</i>	214
7.1.3	<i>The SR Latch</i>	216
7.1.4	<i>The S'R' Latch</i>	219
7.1.5	<i>SR Latch with Enable</i>	222
7.1.6	<i>The D-Latch</i>	223
7.1.7	<i>The D-Flip-Flop</i>	225
7.2	SEQUENTIAL LOGIC TIMING CONSIDERATIONS	229
7.3	COMMON CIRCUITS BASED ON SEQUENTIAL STORAGE DEVICES	230
7.3.1	<i>Toggle Flop Clock Divider</i>	230
7.3.2	<i>Ripple Counter</i>	231
7.3.3	<i>Switch Debouncing</i>	232
7.3.4	<i>Shift Registers</i>	237
7.4	FINITE-STATE MACHINES	238
7.4.1	<i>Describing the Functionality of a FSM</i>	238
7.4.2	<i>Logic Synthesis for a FSM</i>	241
7.4.3	<i>FSM Design Process Overview</i>	248
7.4.4	<i>FSM Design Examples</i>	249
7.5	COUNTERS	256
7.5.1	<i>2-Bit Binary Up Counter</i>	256
7.5.2	<i>2-Bit Binary Up/Down Counter</i>	257
7.5.3	<i>2-Bit Gray Code Up Counter</i>	259
7.5.4	<i>2-Bit Gray Code Up/Down Counter</i>	261
7.5.5	<i>3-Bit One-Hot Up Counter</i>	263
7.5.6	<i>3-Bit One-Hot Up/Down Counter</i>	265

7.6	FINITE-STATE MACHINE'S RESET CONDITION	269
7.7	SEQUENTIAL LOGIC ANALYSIS	270
7.7.1	<i>Finding the State Equations and Output Logic Expressions of a FSM</i>	270
7.7.2	<i>Finding the State Transition Table of a FSM</i>	271
7.7.3	<i>Finding the State Diagram of a FSM</i>	272
7.7.4	<i>Determining the Maximum Clock Frequency of a FSM</i>	273
8:	VERILOG (PART 2)	289
8.1	PROCEDURAL ASSIGNMENT	289
8.1.1	<i>Procedural Blocks</i>	289
8.1.2	<i>Procedural Statements</i>	292
8.1.3	<i>Statement Groups</i>	297
8.1.4	<i>Local Variables</i>	298
8.2	CONDITIONAL PROGRAMMING CONSTRUCTS	298
8.2.1	<i>if-else Statements</i>	298
8.2.2	<i>case Statements</i>	300
8.2.3	<i>casez and casex Statements</i>	301
8.2.4	<i>forever Loops</i>	301
8.2.5	<i>while Loops</i>	302
8.2.6	<i>repeat Loops</i>	303
8.2.7	<i>for loops</i>	303
8.2.8	<i>disable</i>	303
8.3	SYSTEM TASKS	304
8.3.1	<i>Text Output</i>	304
8.3.2	<i>File Input/Output</i>	306
8.3.3	<i>Simulation Control and Monitoring</i>	308
8.4	TEST BENCHES	309
8.4.1	<i>Common Stimulus Generation Techniques</i>	309
8.4.2	<i>Printing Results to the Simulator Transcript</i>	311
8.4.3	<i>Automatic Result Checking</i>	313
8.4.4	<i>Using Loops to Generate Stimulus</i>	314
8.4.5	<i>Using External Files in Test Benches</i>	315
9:	BEHAVIORAL MODELING OF SEQUENTIAL LOGIC	323
9.1	MODELING SEQUENTIAL STORAGE DEVICES IN VERILOG	323
9.1.1	<i>D-Latch</i>	323
9.1.2	<i>D-Flip-Flop</i>	324
9.1.3	<i>D-Flip-Flop with Asynchronous Reset</i>	324
9.1.4	<i>D-Flip-Flop with Asynchronous Reset and Preset</i>	325
9.1.5	<i>D-Flip-Flop with Synchronous Enable</i>	326
9.2	MODELING FINITE-STATE MACHINES IN VERILOG	327
9.2.1	<i>Modeling the States</i>	329
9.2.2	<i>The State Memory Block</i>	329
9.2.3	<i>The Next State Logic Block</i>	329
9.2.4	<i>The Output Logic Block</i>	330
9.2.5	<i>Changing the State Encoding Approach</i>	332

9.3	FSM DESIGN EXAMPLES IN VERILOG	333
9.3.1	Serial Bit Sequence Detector in Verilog	333
9.3.2	Vending Machine Controller in Verilog	336
9.3.3	2-Bit, Binary Up/Down Counter in Verilog	338
9.4	MODELING COUNTERS IN VERILOG	340
9.4.1	Counters in Verilog Using a Single-Procedural Block	340
9.4.2	Counters with Range Checking	341
9.4.3	Counters with Enables in Verilog	342
9.4.4	Counters with Loads	343
9.5	RTL MODELING	344
9.5.1	Modeling Registers in Verilog	345
9.5.2	Registers as Agents on a Data Bus	346
9.5.3	Shift Registers in Verilog	348
10:	MEMORY	355
10.1	MEMORY ARCHITECTURE AND TERMINOLOGY	355
10.1.1	Memory Map Model	355
10.1.2	Volatile Versus Non-volatile Memory	356
10.1.3	Read-Only Versus Read/Write Memory	356
10.1.4	Random Access Versus Sequential Access	356
10.2	NON-VOLATILE MEMORY TECHNOLOGY	357
10.2.1	ROM Architecture	357
10.2.2	Mask Read-Only Memory (MROM)	360
10.2.3	Programmable Read-Only Memory (PROM)	361
10.2.4	Erasable Programmable Read-Only Memory (EPROM)	362
10.2.5	Electrically Erasable Programmable Read-Only Memory (EEPROM)	364
10.2.6	FLASH Memory	365
10.3	VOLATILE MEMORY TECHNOLOGY	365
10.3.1	Static Random-Access Memory (SRAM)	366
10.3.2	Dynamic Random-Access Memory (DRAM)	369
10.4	MODELING MEMORY WITH VERILOG	376
10.4.1	Read-Only Memory in Verilog	376
10.4.2	Read/Write Memory in Verilog	377
11:	PROGRAMMABLE LOGIC	383
11.1	PROGRAMMABLE ARRAYS	383
11.1.1	Programmable Logic Array (PLA)	383
11.1.2	Programmable Array Logic (PAL)	384
11.1.3	Generic Array Logic (GAL)	385
11.1.4	Hard Array Logic (HAL)	386
11.1.5	Complex Programmable Logic Devices (CPLD)	386
11.2	FIELD-PROGRAMMABLE GATE ARRAYS (FPGAs)	387
11.2.1	Configurable Logic Block (or Logic Element)	388
11.2.2	Look-Up Tables (LUTs)	389
11.2.3	Programmable Interconnect Points (PIPs)	392
11.2.4	Input/Output Block (IOB)	393
11.2.5	Configuration Memory	394

12: ARITHMETIC CIRCUITS	397
12.1 ADDITION	397
12.1.1 <i>Half Adders</i>	397
12.1.2 <i>Full Adders</i>	398
12.1.3 <i>Ripple Carry Adder (RCA)</i>	400
12.1.4 <i>Carry Look Ahead Adder (CLA)</i>	402
12.1.5 <i>Adders in Verilog</i>	405
12.2 SUBTRACTION	410
12.3 MULTIPLICATION	413
12.3.1 <i>Unsigned Multiplication</i>	413
12.3.2 <i>A Simple Circuit to Multiply by Powers of Two</i>	416
12.3.3 <i>Signed Multiplication</i>	417
12.4 DIVISION	419
12.4.1 <i>Unsigned Division</i>	419
12.4.2 <i>A Simple Circuit to Divide by Powers of Two</i>	422
12.4.3 <i>Signed Division</i>	423
13: COMPUTER SYSTEM DESIGN	427
13.1 COMPUTER HARDWARE	427
13.1.1 <i>Program Memory</i>	428
13.1.2 <i>Data Memory</i>	428
13.1.3 <i>Input/Output Ports</i>	428
13.1.4 <i>Central Processing Unit</i>	429
13.1.5 <i>A Memory-Mapped System</i>	430
13.2 COMPUTER SOFTWARE	432
13.2.1 <i>Opcodes and Operands</i>	433
13.2.2 <i>Addressing Modes</i>	433
13.2.3 <i>Classes of Instructions</i>	434
13.3 COMPUTER IMPLEMENTATION: AN 8-BIT COMPUTER EXAMPLE	441
13.3.1 <i>Top-Level Block Diagram</i>	441
13.3.2 <i>Instruction Set Design</i>	443
13.3.3 <i>Memory System Implementation</i>	444
13.3.4 <i>CPU Implementation</i>	447
13.4 ARCHITECTURE CONSIDERATIONS	468
13.4.1 <i>Von Neumann Architecture</i>	468
13.4.2 <i>Harvard Architecture</i>	468
APPENDIX A: LIST OF WORKED EXAMPLES	473
APPENDIX B: CONCEPT CHECK SOLUTIONS	479
INDEX	481