
INTRODUCTION TO LOGIC CIRCUITS & LOGIC DESIGN WITH VERILOG

INTRODUCTION TO LOGIC CIRCUITS & LOGIC DESIGN WITH VERILOG

1ST EDITION

Brock J. LaMeres

 Springer

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

ISBN 978-3-319-53882-2 ISBN 978-3-319-53883-9 (eBook)
DOI 10.1007/978-3-319-53883-9

Library of Congress Control Number: 2017932539

© Springer International Publishing AG 2017

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.

Printed on acid-free paper

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

Preface

The purpose of this new 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 (HDLs) 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 & 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. This 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. This 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 this textbook introduces the details of programmable logic, semiconductor memory, and arithmetic circuits. This 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 possess 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 600+ exercise problems and concept check questions for each section tied directly to specific learning outcomes for both formative and summative assessment.

Teaching by Example

With over 200 worked examples, concept checks for each section, 200+ supporting figures, and 600+ exercise 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 this 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

This 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 numbers 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 in 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 request a solution manual that contains a graphic-rich description of the solutions for each of the 600+ exercise problems. Instructors can also receive the Verilog solutions and test benches for each Verilog design exercise. 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.

Acknowledgments

Dr. LaMeres is eternally grateful to his family for their support of this project. To JoAnn, your love and friendship makes everything possible. To Alexis, your kindness and caring brings joy to my heart. To Kylie, your humor and spirit fills me with laughter and pride. Thank you so much.

Dr. LaMeres would also like to thank the 400+ engineering students at Montana State University that helped proof read this book in preparation for the first edition.

Contents

1: INTRODUCTION: ANALOG VS. DIGITAL	1
1.1 DIFFERENCES BETWEEN ANALOG AND DIGITAL SYSTEMS	1
1.2 ADVANTAGES OF DIGITAL SYSTEMS OVER ANALOG SYSTEMS	2
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>	17
2.3 BINARY ARITHMETIC	21
2.3.1 <i>Addition (Carries)</i>	21
2.3.2 <i>Subtraction (Borrows)</i>	22
2.4 UNSIGNED AND SIGNED NUMBERS	23
2.4.1 <i>Unsigned Numbers</i>	24
2.4.2 <i>Signed Numbers</i>	24
3: DIGITAL CIRCUITRY AND INTERFACING	37
3.1 BASIC GATES	37
3.1.1 <i>Describing the Operation of a Logic Circuit</i>	37
3.1.2 <i>The Buffer</i>	39
3.1.3 <i>The Inverter</i>	40
3.1.4 <i>The AND Gate</i>	40
3.1.5 <i>The NAND Gate</i>	41
3.1.6 <i>The OR Gate</i>	41
3.1.7 <i>The NOR Gate</i>	41
3.1.8 <i>The XOR Gate</i>	42
3.1.9 <i>The XNOR Gate</i>	43
3.2 DIGITAL CIRCUIT OPERATION	44
3.2.1 <i>Logic Levels</i>	44
3.2.2 <i>Output DC Specifications</i>	45
3.2.3 <i>Input DC Specifications</i>	46
3.2.4 <i>Noise Margins</i>	47
3.2.5 <i>Power Supplies</i>	48
3.2.6 <i>Switching Characteristics</i>	51
3.2.7 <i>Data Sheets</i>	51

3.3	LOGIC FAMILIES	56
3.3.1	<i>Complementary Metal Oxide Semiconductors (CMOS)</i>	56
3.3.2	<i>Transistor-Transistor Logic (TTL)</i>	65
3.3.3	<i>The 7400 Series Logic Families</i>	67
3.4	DRIVING LOADS	71
3.4.1	<i>Driving Other Gates</i>	71
3.4.2	<i>Driving Resistive Loads</i>	73
3.4.3	<i>Driving LEDs</i>	75
4:	COMBINATIONAL LOGIC DESIGN	81
4.1	BOOLEAN ALGEBRA	81
4.1.1	<i>Operations</i>	82
4.1.2	<i>Axioms</i>	82
4.1.3	<i>Theorems</i>	83
4.1.4	<i>Functionally Complete Operation Sets</i>	98
4.2	COMBINATIONAL LOGIC ANALYSIS	99
4.2.1	<i>Finding the Logic Expression from a Logic Diagram</i>	99
4.2.2	<i>Finding the Truth Table from a Logic Diagram</i>	100
4.2.3	<i>Timing Analysis of a Combinational Logic Circuit</i>	101
4.3	COMBINATIONAL LOGIC SYNTHESIS	103
4.3.1	<i>Canonical Sum of Products</i>	103
4.3.2	<i>The Minterm List (Σ)</i>	104
4.3.3	<i>Canonical Product of Sums (POS)</i>	106
4.3.4	<i>The Maxterm List (Π)</i>	108
4.3.5	<i>Minterm and Maxterm List Equivalence</i>	110
4.4	LOGIC MINIMIZATION	112
4.4.1	<i>Algebraic Minimization</i>	112
4.4.2	<i>Minimization Using Karnaugh Maps</i>	113
4.4.3	<i>Don't Cares</i>	125
4.4.4	<i>Using XOR Gates</i>	126
4.5	TIMING HAZARDS & GLITCHES	129
5:	VERILOG (PART 1)	141
5.1	HISTORY OF HARDWARE DESCRIPTION LANGUAGES	142
5.2	HDL ABSTRACTION	145
5.3	THE MODERN DIGITAL DESIGN FLOW	149
5.4	VERILOG CONSTRUCTS	152
5.4.1	<i>Data Types</i>	153
5.4.2	<i>The Module</i>	156
5.4.3	<i>Verilog Operators</i>	159
5.5	MODELING CONCURRENT FUNCTIONALITY IN VERILOG	164
5.5.1	<i>Continuous Assignment</i>	164
5.5.2	<i>Continuous Assignment with Logical Operators</i>	164
5.5.3	<i>Continuous Assignment with Conditional Operators</i>	165
5.5.4	<i>Continuous Assignment with Delay</i>	167

5.6	STRUCTURAL DESIGN AND HIERARCHY	170
5.6.1	<i>Lower-Level Module Instantiation</i>	170
5.6.2	<i>Gate Level Primitives</i>	172
5.6.3	<i>User-Defined Primitives</i>	173
5.6.4	<i>Adding Delay to Primitives</i>	174
5.7	OVERVIEW OF SIMULATION TEST BENCHES	175
6:	MSI LOGIC	181
6.1	DECODERS	181
6.1.1	<i>Example: One-Hot Decoder</i>	181
6.1.2	<i>Example: 7-Segment Display Decoder</i>	184
6.2	ENCODERS	188
6.2.1	<i>Example: One-Hot Binary Encoder</i>	188
6.3	MULTIPLEXERS	190
6.4	DEMULTIPLEXERS	193
7:	SEQUENTIAL LOGIC DESIGN	199
7.1	SEQUENTIAL LOGIC STORAGE DEVICES	199
7.1.1	<i>The Cross-Coupled Inverter Pair</i>	199
7.1.2	<i>Metastability</i>	200
7.1.3	<i>The SR Latch</i>	202
7.1.4	<i>The S'R' Latch</i>	205
7.1.5	<i>SR Latch with Enable</i>	208
7.1.6	<i>The D-Latch</i>	209
7.1.7	<i>The D-Flip-Flop</i>	211
7.2	SEQUENTIAL LOGIC TIMING CONSIDERATIONS	214
7.3	COMMON CIRCUITS BASED ON SEQUENTIAL STORAGE DEVICES	216
7.3.1	<i>Toggle Flop Clock Divider</i>	216
7.3.2	<i>Ripple Counter</i>	217
7.3.3	<i>Switch Debouncing</i>	217
7.3.4	<i>Shift Registers</i>	221
7.4	FINITE STATE MACHINES	223
7.4.1	<i>Describing the Functionality of a FSM</i>	223
7.4.2	<i>Logic Synthesis for a FSM</i>	225
7.4.3	<i>FSM Design Process Overview</i>	232
7.4.4	<i>FSM Design Examples</i>	233
7.5	COUNTERS	241
7.5.1	<i>2-Bit Binary Up Counter</i>	241
7.5.2	<i>2-Bit Binary Up/Down Counter</i>	242
7.5.3	<i>2-Bit Gray Code Up Counter</i>	245
7.5.4	<i>2-Bit Gray Code Up/Down Counter</i>	247
7.5.5	<i>3-Bit One-Hot Up Counter</i>	249
7.5.6	<i>3-Bit One-Hot Up/Down Counter</i>	250
7.6	FINITE STATE MACHINE'S RESET CONDITION	254
7.7	SEQUENTIAL LOGIC ANALYSIS	255
7.7.1	<i>Finding the State Equations and Output Logic Expressions of a FSM</i>	255

7.7.2	<i>Finding the State Transition Table of a FSM</i>	256
7.7.3	<i>Finding the State Diagram of a FSM</i>	257
7.7.4	<i>Determining the Maximum Clock Frequency of a FSM</i>	258
8:	VERILOG (PART 2)	271
8.1	PROCEDURAL ASSIGNMENT	271
8.1.1	<i>Procedural Blocks</i>	271
8.1.2	<i>Procedural Statements</i>	274
8.1.3	<i>Statement Groups</i>	279
8.1.4	<i>Local Variables</i>	279
8.2	CONDITIONAL PROGRAMMING CONSTRUCTS	280
8.2.1	<i>if-else Statements</i>	280
8.2.2	<i>case Statements</i>	281
8.2.3	<i>casez and casex Statements</i>	283
8.2.4	<i>forever Loops</i>	283
8.2.5	<i>while Loops</i>	283
8.2.6	<i>repeat Loops</i>	284
8.2.7	<i>for Loops</i>	284
8.2.8	<i>disable</i>	285
8.3	SYSTEM TASKS	286
8.3.1	<i>Text Output</i>	286
8.3.2	<i>File Input/Output</i>	287
8.3.3	<i>Simulation Control and Monitoring</i>	289
8.4	TEST BENCHES	290
8.4.1	<i>Common Stimulus Generation Techniques</i>	291
8.4.2	<i>Printing Results to the Simulator Transcript</i>	292
8.4.3	<i>Automatic Result Checking</i>	293
8.4.4	<i>Using Loops to Generate Stimulus</i>	295
8.4.5	<i>Using External Files in Test Benches</i>	296
9:	BEHAVIORAL MODELING OF SEQUENTIAL LOGIC	303
9.1	MODELING SEQUENTIAL STORAGE DEVICES IN VERILOG	303
9.1.1	<i>D-Latch</i>	303
9.1.2	<i>D-Flip-Flop</i>	304
9.1.3	<i>D-Flip-Flop with Asynchronous Reset</i>	304
9.1.4	<i>D-Flip-Flop with Asynchronous Reset and Preset</i>	305
9.1.5	<i>D-Flip-Flop with Synchronous Enable</i>	306
9.2	MODELING FINITE STATE MACHINES IN VERILOG	307
9.2.1	<i>Modeling the States</i>	309
9.2.2	<i>The State Memory Block</i>	309
9.2.3	<i>The Next State Logic Block</i>	309
9.2.4	<i>The Output Logic Block</i>	310
9.2.5	<i>Changing the State Encoding Approach</i>	312
9.3	FSM DESIGN EXAMPLES IN VERILOG	313
9.3.1	<i>Serial Bit Sequence Detector in Verilog</i>	313
9.3.2	<i>Vending Machine Controller in Verilog</i>	315
9.3.3	<i>2-Bit, Binary Up/Down Counter in Verilog</i>	317

9.4	MODELING COUNTERS IN VERILOG	319
9.4.1	<i>Counters in Verilog Using a Single Procedural Block</i>	319
9.4.2	<i>Counters with Range Checking</i>	320
9.4.3	<i>Counters with Enables in Verilog</i>	320
9.4.4	<i>Counters with Loads</i>	321
9.5	RTL MODELING	322
9.5.1	<i>Modeling Registers in Verilog</i>	322
9.5.2	<i>Registers as Agents on a Data Bus</i>	323
9.5.3	<i>Shift Registers in Verilog</i>	325
10:	MEMORY	331
10.1	MEMORY ARCHITECTURE AND TERMINOLOGY	331
10.1.1	<i>Memory Map Model</i>	331
10.1.2	<i>Volatile Versus Non-volatile Memory</i>	332
10.1.3	<i>Read Only Versus Read/Write Memory</i>	332
10.1.4	<i>Random Access Versus Sequential Access</i>	332
10.2	NON-VOLATILE MEMORY TECHNOLOGY	333
10.2.1	<i>ROM Architecture</i>	333
10.2.2	<i>Mask Read Only Memory (MROM)</i>	336
10.2.3	<i>Programmable Read Only Memory (PROM)</i>	337
10.2.4	<i>Erasable Programmable Read Only Memory (EPROM)</i>	338
10.2.5	<i>Electrically Erasable Programmable Read Only Memory (EEPROM)</i>	340
10.2.6	<i>FLASH Memory</i>	341
10.3	VOLATILE MEMORY TECHNOLOGY	342
10.3.1	<i>Static Random Access Memory (SRAM)</i>	342
10.3.2	<i>Dynamic Random Access Memory (DRAM)</i>	345
10.4	MODELING MEMORY WITH VERILOG	352
10.4.1	<i>Read-Only Memory in Verilog</i>	352
10.4.2	<i>Read/Write Memory in Verilog</i>	353
11:	PROGRAMMABLE LOGIC	359
11.1	PROGRAMMABLE ARRAYS	359
11.1.1	<i>Programmable Logic Array (PLA)</i>	359
11.1.2	<i>Programmable Array Logic (PAL)</i>	360
11.1.3	<i>Generic Array Logic (GAL)</i>	361
11.1.4	<i>Hard Array Logic (HAL)</i>	362
11.1.5	<i>Complex Programmable Logic Devices (CPLD)</i>	362
11.2	FIELD PROGRAMMABLE GATE ARRAYS (FPGAs)	363
11.2.1	<i>Configurable Logic Block (or Logic Element)</i>	364
11.2.2	<i>Look-Up Tables (LUTs)</i>	365
11.2.3	<i>Programmable Interconnect Points (PIPs)</i>	368
11.2.4	<i>Input/Output Block (IOBs)</i>	369
11.2.5	<i>Configuration Memory</i>	370

12: ARITHMETIC CIRCUITS	373
12.1 ADDITION	373
12.1.1 <i>Half Adders</i>	373
12.1.2 <i>Full Adders</i>	374
12.1.3 <i>Ripple Carry Adder (RCA)</i>	376
12.1.4 <i>Carry Look Ahead Adder (CLA)</i>	378
12.1.5 <i>Adders in Verilog</i>	381
12.2 SUBTRACTION	386
12.3 MULTIPLICATION	389
12.3.1 <i>Unsigned Multiplication</i>	389
12.3.2 <i>A Simple Circuit to Multiply by Powers of Two</i>	392
12.3.3 <i>Signed Multiplication</i>	393
12.4 DIVISION	395
12.4.1 <i>Unsigned Division</i>	395
12.4.2 <i>A Simple Circuit to Divide by Powers of Two</i>	398
12.4.3 <i>Signed Division</i>	399
13: COMPUTER SYSTEM DESIGN	403
13.1 COMPUTER HARDWARE	403
13.1.1 <i>Program Memory</i>	404
13.1.2 <i>Data Memory</i>	404
13.1.3 <i>Input/Output Ports</i>	404
13.1.4 <i>Central Processing Unit</i>	405
13.1.5 <i>A Memory Mapped System</i>	406
13.2 COMPUTER SOFTWARE	408
13.2.1 <i>Opcodes and Operands</i>	409
13.2.2 <i>Addressing Modes</i>	409
13.2.3 <i>Classes of Instructions</i>	410
13.3 COMPUTER IMPLEMENTATION – AN 8-BIT COMPUTER EXAMPLE	417
13.3.1 <i>Top Level Block Diagram</i>	417
13.3.2 <i>Instruction Set Design</i>	418
13.3.3 <i>Memory System Implementation</i>	419
13.3.4 <i>CPU Implementation</i>	423
13.4 ARCHITECTURE CONSIDERATIONS	444
13.4.1 <i>Von Neumann Architecture</i>	444
13.4.2 <i>Harvard Architecture</i>	444
APPENDIX A: LIST OF WORKED EXAMPLES	449
INDEX	455