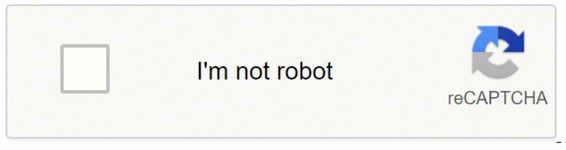

Continue

Fundamentals of logic design pdf

www.allitebooks.com Fundamentals of Logic Design SEVENTH EDITION Charles H. Roth, Jr. University of Texas at Austin Larry L. Kinney University of Minnesota, Twin Cities $XVWUDOLDä%UD]LOä-DSDQä.RUHDä0H[LFRä6LQJDSRUHä6SDLQä8QLWHG.LQJGRPä8QLWHG6WDWHV www.allitebooks.com k20142010DQG2004&HQJDJH/HDUQLQJ Fundamentals of Logic Design, Seventh Edition $//5,*+765(6(59('1RSDUWRIWKLVZRUNFRYHUHGE\WKH FRS\ULJKWKHUHLQPD\EHUHSURGXFHGWUDQVPLWWHGVWRUHGRUXVHG LQDQ\IRUPRUE\DQ\PHDQVJUDSKLFHOHFWURQLFRUPHFKDQLFDO LQFOXGLQJEXWQRWOLPLWHGWRSKRWRFRS\LQJUHFRUGLQJVFDQQLQJ GLJLWL]LQJWDSLQJZHEGLVWULEXWLRQLQIRUPDWLRQQHWZRUNVRU LQIRUPDWLRQVWRUDJHDQGUHWULHYDOV\VWHPVH[FHSWDVSHUPLWWHGXQGHU 6HFWLRQ107RU108RIWKH19768QLWHG6WDWHV&RS\ULJKW$FWZLWKRXW WKHSULRUZULWWHQSHUPLVVLRQRIWKHSXEOLVKHU Charles H. Roth, Jr. and Larry L. Kinney 3XEOLVKHU*OREDO(QJLQHHULQJ7LP$QGHUVRQ 6HQLRU'HYHORSPHQWDO(GLWRU+LOGD*RZDQV (GLWRULDO$VVLVWDQW7DQ\D$OWLHUL 6HQLRU0DUNHWLQJ0DQDJHU.DWH,DQQRWWL 0HGLD(GLWRU&KULV9DOHQWLQH)RUSURGXFWLQIRUPDWLRQDQGWHFKQRORJ\DVVLVWDQFHFRQWDFWXVDW Cengage Learning Customer & Sales Support, 1-800-354-9706 &RQWHQW3URMHFW0DQDJHU-HQQLIHU=LHJOHU 3URGXFWLRQ6HUYLFH53.(GLWRULDO6HUYLFHV &RS\HGLWRU3DWULFLD'DO\ )RUSHUPLVVLRQWRXVHPDWHULDOIURPWKLVWH[WRUSURGXFWVXEPLWDOO UHTXHVWVRQOLQHDWwww.cengage.com/permissions)XUWKHUSHUPLVVLRQVTXHVWLRQVFDQEHHPDLOHGWR 3URRIUHDGHU0DUWKD0F0DVWHU ,QGH[HU6KHOO*HUJHU.QHFKWO &RPSRVLWRUQGLDFUL7HFK 6HQLRU$UW'LUHFWRU0LFKHOOH.XQNOHU /LEUDU\RI&RQJUHVV&RQWURO1XPEHU2012952056 ,QWHUQDO'HVLJQHU&DUPHOD3HULDQHD ,6%113978-1-133-62847-7 ,6%1101133628478 &RYHU'HVLJQHU5RVH$OFRUQ &RYHU,PDJHk.XGU\DVKND6KXWWHUVWRFNFRP kWRYRYDQ6KXWWHUVWRFNFRP 5LJKWV$FTXLVLWLRQV6SHFLDOLVW$PEHU+RVHD 7H[WDQG,PDJH3HUPLVVLRQV5HVHDUFKHU .ULVWLLQD3DXO 6HQLRU0DQXIDFWXULQJ3ODQQHU'RXJ:LONH Cengage Learning 200)LUVW6WDPIRUG3ODFH6XLWH400 6WDPIRUG&706902 86$ &HQJDJH/HDUQLQJLVDOHDGLQJSURYLGHURIFXVWRPL]HGOHDUQLQJ VROXWLRQVZLWKRėFHORFDWLRQVDURXQGWKHJOREHLQFOXGLQJ6LQJDSRUH WKH8QLWHG.LQJGRP$XVWUDOLD0H[LFR%UD]LODQG-DSDQ/RFDWH\RXU ORFDORėFHDWinternational.cengage.com/region &HQJDJH/HDUQLQJSURGXFWVDUHUHSUHVHQWHGLQ&DQDGDE\ 1HOVRQ(GXFDWLRQ/WG)RU\RXUFRXUVHDQGOHDUQLQJVROXWLRQVYLVLW www.cengage.com/engineering 3XUFKDVHDQ\RIRXUSURGXFWVDW\RXUORFDOFROOHJHVWRUHRUDWRXU SUHIHUUHGRQOLQHVWRUHwww.cengagebrain.com ([FHSWZKHUHRWKHUZLVHQRWHGDOOFRQWHQWLVk&HQJDJH/HDUQLQJ2014 Printed in the United States of America 1 2 3 4 5 6 7 16 15 14 13 www.allitebooks.com Dedication Dedicated to the memory of Karen Kinney and our daughters, Laurie and Kristina. —Larry Kinney www.allitebooks.com Brief Contents 1 2 3 Introduction Number Systems and Conversion 4 5 6 Applications of Boolean Algebra Minterm and Maxterm Expansions 7 Multi-Level Gate Circuits NAND and NOR Gates 193 8 Combinational Circuit Design and Simulation Using Gates 225 9 Multiplexers, Decoders, and Programmable Logic Devices 252 Boolean Algebra 29 Boolean Algebra (Continued) Karnaugh Maps 1 60 87 123 Quine-McCluskey Method www.allitebooks.com 167 viii Brief Contents 10 11 12 13 14 15 16 17 18 19 20 Introduction to VHDL Latches and Flip-Flops 294 331 Registers and Counters 370 Analysis of Clocked Sequential Circuits Derivation of State Graphs and Tables 412 453 Reduction of State Tables State Assignment 497 Sequential Circuit Design VHDL for Sequential Logic 545 585 Circuits for Arithmetic Operations 626 State Machine Design with SM Charts VHDL for Digital System Design Appendices 713 www.allitebooks.com 684 660 Contents Preface xvii How to Use This Book for Self-Study About the Authors xxiii xxii Unit 1 Introduction Number Systems and Conversion 1.1 1.2 1.3 1.4 1.5 Objectives 1 Study Guide 2 Digital Systems and Switching Circuits 6 Number Systems and Conversion 8 Binary Arithmetic 12 Representation of Negative Numbers 16 Sign and Magnitude Numbers 16 2’s Complement Numbers 16 Addition of 2’s Complement Numbers 17 1’s Complement Numbers 19 Addition of 1’s Complement Numbers 19 Binary Codes 21 Problems 24 Unit 2 Boolean Algebra 2.1 2.2 2.3 1 29 Objectives 29 Study Guide 30 Introduction 36 Basic Operations 37 Boolean Expressions and Truth Tables 39 ix www.allitebooks.com x Contents 2.4 2.5 2.6 2.7 2.8 Basic Theorems 41 Commutative, Associative, Distributive, and DeMorgan’s Laws 43 Simplification Theorems 46 Multiplying Out and Factoring 49 Complementing Boolean Expressions 52 Problems 53 Unit 3 Boolean Algebra (Continued) 3.1 3.2 3.3 3.4 3.5 60 Objectives 60 Study Guide 61 Multiplying Out and Factoring Expressions 66 Exclusive-OR and Equivalence Operations 68 The Consensus Theorem 70 Algebraic Simplification of Switching Expressions Proving Validity of an Equation 74 Programmed Exercises 77 Problems 82 Unit 4 Applications of Boolean Algebra Minterm and Maxterm Expansions 4.1 4.2 4.3 4.4 4.5 4.6 4.7 87 Objectives 87 Study Guide 88 Conversion of English Sentences to Boolean Equations Combinational Logic Design Using a Truth Table 96 Minterm and Maxterm Expansions 97 General Minterm and Maxterm Expansions 100 Incompletely Specified Functions 103 Examples of Truth Table Construction 104 Design of Binary Adders and Subtracters 108 Problems 114 Unit 5 Karnaugh Maps 5.1 5.2 72 123 Objectives 123 Study Guide 124 Minimum Forms of Switching Functions Two- and Three-Variable Karnaugh Maps www.allitebooks.com 134 136 94 Contents 5.3 5.4 5.5 5.6 5.7 Four-Variable Karnaugh Maps 141 Determination of Minimum Expressions Using Essential Prime Implicants 144 Five-Variable Karnaugh Maps 149 Other Uses of Karnaugh Maps 152 Other Forms of Karnaugh Maps 153 Programmed Exercises 154 Problems 159 Unit 6 Quine-McCluskey Method 6.1 6.2 6.3 6.4 6.5 6.6 167 Objectives 167 Study Guide 168 Determination of Prime Implicants 173 The Prime Implicant Chart 176 Petrick’s Method 179 Simplification of Incompletely Specified Functions Simplification Using Map-Entered Variables 182 Conclusion 184 Programmed Exercise 185 Problems 189 181 Unit 7 Multi-Level Gate Circuits NAND and NOR Gates 193 7.1 7.2 7.3 7.4 7.5 7.6 7.7 Objectives 193 Study Guide 194 Multi-Level Gate Circuits 199 NAND and NOR Gates 204 Design of Two-Level NAND- and NOR-Gate Circuits 206 Design of Multi-Level NAND- and NOR-Gate Circuits 209 Circuit Conversion Using Alternative Gate Symbols 210 Design of Two-Level, Multiple-Output Circuits 214 Determination of Essential Prime Implicants for Multiple-Output Realization 216 Multiple-Output NAND- and NOR-Gate Circuits 217 Problems 218 www.allitebooks.com xi xii Contents Unit 8 Combinational Circuit Design and Simulation Using Gates 225 8.1 8.2 8.3 8.4 8.5 Objectives 225 Study Guide 226 Review of Combinational Circuit Design 229 Design of Circuits with Limited Gate Fan-In 230 Gate Delays and Timing Diagrams 232 Hazards in Combinational Logic 234 Simulation and Testing of Logic Circuits 240 Problems 243 Design Problems 246 Seven-Segment Indicator 246 Unit 9 Multiplexers, Decoders, and Programmable Logic Devices 252 9.1 9.2 9.3 9.4 9.5 9.6 9.7 9.8 Objectives 252 Study Guide 253 Introduction 260 Multiplexers 261 Three-State Buffers 265 Decoders and Encoders 268 Read-Only Memories 271 Programmable Logic Devices 275 Programmable Logic Arrays 275 Programmable Array Logic 278 Complex Programmable Logic Devices 280 Field-Programmable Gate Arrays 282 Decomposition of Switching Functions 283 Problems 286 Unit 10 Introduction to VHDL 294 Objectives 294 Study Guide 295 10.1 VHDL Description of Combinational Circuits 10.2 VHDL Models for Multiplexers 304 10.3 VHDL Modules 306 Four-Bit Full Adder 308 www.allitebooks.com 299 Contents 10.4 10.5 10.6 10.7 10.8 10.9 Signals and Constants 311 Arrays 312 VHDL Operators 315 Packages and Libraries 316 IEEE Standard Logic 318 Compilation and Simulation of VHDL Code Problems 322 Design Problems 327 Unit 11 Latches and Flip-Flops 11.1 11.2 11.3 11.4 11.5 11.6 11.7 11.8 11.9 11.10 12.2 12.3 12.4 12.5 12.6 331 Objectives 331 Study Guide 332 Introduction 336 Set-Reset Latch 338 Gated Latches 342 Edge-Triggered D Flip-Flop 346 S-R Flip-Flop 349 J-K Flip-Flop 350 T Flip-Flop 351 Flip-Flops with Additional Inputs 352 Asynchronous Sequential Circuits 354 Summary 357 Problems 358 Programmed Exercise 367 Unit 12 Registers and Counters 12.1 321 370 Objectives 370 Study Guide 371 Registers and Register Transfers 376 Parallel Adder with Accumulator 378 Shift Registers 380 Design of Binary Counters 384 Counters for Other Sequences 389 Counter Design Using D Flip-Flops 393 Counter Design Using S-R and J-K Flip-Flops 395 Derivation of Flip-Flop Input Equations—Summary Problems 402 398 xiii xiv Contents Unit 13 Analysis of Clocked Sequential Circuits 13.1 13.2 13.3 13.4 Objectives 412 Study Guide 413 A Sequential Parity Checker 419 Analysis by Signal Tracing and Timing Charts 421 State Tables and Graphs 425 Construction and Interpretation of Timing Charts 430 General Models for Sequential Circuits 432 Programmed Exercise 436 Problems 441 Unit 14 Derivation of State Graphs and Tables 14.1 14.2 14.3 14.4 14.5 14.6 412 453 Objectives 453 Study Guide 454 Design of a Sequence Detector 457 More Complex Design Problems 463 Guidelines for Construction of State Graphs Serial Data Code Conversion 473 Alphanumeric State Graph Notation 476 Incompletely Specified State Tables 478 Programmed Exercises 480 Problems 486 467 Unit 15 Reduction of State Tables State Assignment 497 15.1 15.2 15.3 15.4 15.5 15.6 15.7 15.8 15.9 Objectives 497 Study Guide 498 Elimination of Redundant States 505 Equivalent States 507 Determination of State Equivalence Using an Implication Table 509 Equivalent Sequential Circuits 512 Reducing Incompletely Specified State Tables 514 Derivation of Flip-Flop Input Equations 517 Equivalent State Assignments 519 Guidelines for State Assignment 523 Using a One-Hot State Assignment 528 Problems 531 Contents Unit 16 Sequential Circuit Design 16.1 16.2 16.3 16.4 16.5 16.6 16.7 16.8 17.1 17.2 17.3 17.4 17.5 17.6 545 Objectives 545 Study Guide 546 Summary of Design Procedure for Sequential Circuits Design Example—Code Converter 549 Design of Iterative Circuits 553 Design of a Comparator 553 Design of Sequential Circuits Using ROMs and PLAs Sequential Circuit Design Using CPLDs 559 Sequential Circuit Design Using FPGAs 563 Simulation and Testing of Sequential Circuits 565 Overview of Computer-Aided Design 570 Design Problems 572 Additional Problems 578 Unit 17 VHDL for Sequential Logic xv 548 556 585 Objectives 585 Study Guide 586 Modeling Flip-Flops Using VHDL Processes 590 Modeling Registers and Counters Using VHDL Processes 594 Modeling Combinational Logic Using VHDL Processes Modeling a Sequential Machine 601 Synthesis of VHDL Code 608 More About Processes and Sequential Statements 611 Problems 613 Simulation Problems 624 Unit 18 Circuits for Arithmetic Operations 626 Objectives 626 Study Guide 627 18.1 Serial Adder with Accumulator 629 18.2 Design of a Binary Multiplier 633 18.3 Design of a Binary Divider 637 Programmed Exercises 644 Problems 648 599 xvi Contents Unit 19 State Machine Design with SM Charts 660 Objectives 660 Study Guide 661 19.1 State Machine Charts 662 19.2 Derivation of SM Charts 667 19.3 Realization of SM Charts 672 Problems 677 Unit 20 VHDL for Digital System Design 20.1 20.2 20.3 20.4 20.5 Objectives 684 Study Guide 685 VHDL Code for a Serial Adder 688 VHDL Code for a Binary Multiplier 690 VHDL Code for a Binary Divider 700 VHDL Code for a Dice Game Simulator 702 Concluding Remarks 705 Problems 706 Lab Design Problems 709 A Appendices A B C D E 684 713 MOS and CMOS Logic 713 VHDL Language Summary 719 Tips for Writing Synthesizable VHDL Code 724 Proofs of Theorems 727 Answers to Selected Study Guide Questions and Problems References 785 Index 786 Description of the CD 792 729 Preface Purpose of the Text This text is written for a first course in the logic design of digital systems. It is written on the premise that the student should understand and learn thoroughly certain fundamental concepts in a first course. Examples of such fundamental concepts are the use of Boolean algebra to describe the signals and interconnections in a logic circuit, use of systematic techniques for simplification of a logic circuit, interconnection of simple components to perform a more complex logic function, analysis of a sequential logic circuit in terms of timing charts or state graphs, and use of a control circuit to control the sequence of events in a digital system. The text attempts to achieve a balance between theory and application. For this reason, the text does not overemphasize the mathematics of switching theory; however, it does present the theory that is necessary for understanding the fundamental concepts of logic design. After completing this text, the student should be prepared for a more advanced digital systems design course that stresses more intuitive concepts like the development of algorithms for digital processes, partitioning of digital systems into subsystems, and implementation of digital systems using currently available hardware. Alternatively, the student should be prepared to go on to a more advanced course in switching theory that further develops the theoretical concepts that have been introduced here. Contents of the Text After studying this text, students should be able to apply switching theory to the solution of logic design problems. They will learn both the basic theory of switching circuits and how to apply it. After a brief introduction to number systems, they will study switching algebra, a special case of Boolean algebra, which is the basic mathematical tool needed to analyze and synthesize an important class of switching xvii xviii Preface circuits. Starting from a problem statement, they will learn to design circuits of logic gates that have a specified relationship between signals at the input and output terminals. Then they will study the logical properties of flip-flops, which serve as memory devices in sequential switching circuits. By combining flip-flops with circuits of logic gates, they will learn to design counters, adders, sequence detectors, and similar circuits. They will also study the VHDL hardware description language and its application to the design of combinational logic, sequential logic, and simple digital systems. As integrated circuit technology continues to improve to allow more components on a chip, digital systems continue to grow in complexity. Design of such complex systems is facilitated by the use of a hardware description language such as VHDL. This text introduces the use of VHDL in logic design and emphasizes the relationship between VHDL statements and the corresponding digital hardware. VHDL allows digital hardware to be described and simulated at a higher level before it is implemented with logic components. Computer programs for synthesis can convert a VHDL description of a digital system to a corresponding set of logic components and their interconnections. Even though use of such computer-aided design tools helps to automate the logic design process, we believe that it is important to understand the underlying logic components and their timing before writing VHDL code. By first implementing the digital logic manually, students can more fully appreciate the power and limitations of VHDL. Although the technology used to implement digital systems has changed significantly since the first edition of this text was published, the fundamental principles of logic design have not. Truth tables and state tables still are used to specify the behavior of logic circuits, and Boolean algebra is still a basic mathematical tool for logic design. Even when programmable logic devices (PLDs) are used instead of individual gates and flip-flops, reduction of logic equations is still desirable in order to fit the equations into smaller PLDs. Making a good state assignment is still desirable, because without a good assignment, the logic equations may require larger PLDs. Strengths of the Text Although many texts are available in the areas of switching theory and logic design, this text is designed so that it can be used in either a standard lecture course or in a self-paced course. In addition to the standard reading material and problems, study guides and other aids for self-study are included in the text. The content of the text is divided into 20 study units. These units form a logical sequence so that mastery of the material in one unit is generally a prerequisite to the study of succeeding units. Each unit consists of four parts. First, a list of objectives states precisely what you are expected to learn by studying the unit. Next, the study guide contains reading assignments and study questions. As you work through the unit, you should write out the answers to these study questions. The text material and problem set that follow Preface xix are similar to a conventional textbook. When you complete a unit, you should review the objectives and make sure that you have met them. Each of the units has undergone extensive class testing in a self-paced environment and has been revised based on student feedback. The study units are divided into three main groups. The first 9 units treat Boolean algebra and the design of combinational logic circuits. Units 11 through 16, 18 and 19 are mainly concerned with the analysis and design of clocked sequential logic circuits, including circuits for arithmetic operations. Units 10, 17, and 20 introduce the VHDL hardware description language and its application to logic design. The text is suitable for both computer science and engineering students. Material relating to circuit aspects of logic gates is contained in Appendix A so that this material can conveniently be omitted by computer science students or other students with no background in electronic circuits. The text is organized so that Unit 6 on the Quine-McCluskey procedure may be omitted without loss of continuity. The three units on VHDL can be studied in the normal sequence, studied together after the other units, or omitted entirely. Supplements and Resources This book comes with support materials for both the instructor and the student. The supplements are housed on the book’s companion website. To access the additional course materials, please visit www.cengagebrain.com. At the cengagebrain.com home page, search for the ISBN of your title (from the back cover of your book) using the search box at the top of the page. This will take you to the product page where these resources can be found. Instructor Resources An instructor’s solution manual (ISM) is available that includes suggestions for using the text in a standard or self-paced course, quizzes on each of the units, and suggestions for laboratory equipment and procedures. The instructor’s manual also contains solutions to problems, to unit quizzes, and to lab exercises. The ISM is available in both print and digital formats. The digital version is available to registered instructors at the publisher’s website. This website also includes both a full set of PowerPoint slides containing all graphical images and tables in the text, and a set of Lecture Builder PowerPoint slides of all equations and example problems. Student Resources Since the computer plays an important role in the logic design process, integration of computer usage into the first logic design course is very important. A computer-aided logic design program, called LogicAid, is included on the CD that accompanies this xx Preface text. LogicAid allows the student to easily derive simplified logic equations from minterms, truth tables, and state tables. This relieves the student of some of the more tedious computations and permits the solution of more complex design problems in a shorter time. LogicAid also provides tutorial help for Karnaugh maps and derivation of state graphs. Several of the units include simulation or laboratory exercises. These exercises provide an opportunity to design a logic circuit and then test its operation. The SimUaid logic simulator, also available on the book’s accompanying CD, may be used to verify the logic designs. The lab equipment required for testing either can be a breadboard with integrated circuit flip-flops and logic gates or a circuit board with a programmable logic device. If such equipment is not available, the lab exercises can be simulated with SimUaid or just assigned as design problems. This is especially important for Units 8, 16, and 20 because the comprehensive design problems in these units help to review and tie together the material in several of the preceding units. The DirectVHDL software on the CD provides a quick way to check and simulate VHDL descriptions of hardware. This software checks the syntax of the VHDL code as it is typed in so that most syntax errors can be corrected before the simulation phase. Changes from Previous Editions The text has evolved considerably since the fifth edition. Programmable logic and the VHDL hardware description language were added, and an emphasis was placed on the role of simulation and computer-aided design of logic circuits. The discussion of VHDL, hazards, latches and one-hot state assignments was expanded. Numerous problems were added. Several additional changes have been made for the seventh edition. The discussion of number systems was reorganized so that one’s complement number systems can be easily omitted. In the unit on Boolean algebra, the laws of switching algebra are first derived using switch networks and truth tables; these are used to define Boolean algebra and, then, further theorems of Boolean algebra are derived that are useful in simplifying switching algebra expressions. The discussion of adders is expanded to include carry-lookahead adders. Alternative implementations of multiplexers are included and also a discussion of active high and active low signals. Other types of gated latches are discussed, and a brief introduction to asynchronous sequential circuits is included. There is more discussion of incompletely specified state tables and how they may occur, and reducing incompletely specified state tables is briefly discussed. Problems have been added throughout the book with an emphasis on more challenging problems than the typical exercises. In addition, the logic design and simulation software that accompanies the text has been updated and improved. Preface xxi Acknowledgments To be effective, a book designed for self-study cannot simply be written. It must be tested and revised many times to achieve its goals. We wish to express our appreciation to the many professors, proctors, and students who participated in this process. Special thanks go to Dr. David Brown, who helped teach the self-paced course, and who made many helpful suggestions for improving the fifth edition. Special thanks to graduate teaching assistant, Mark Story, who developed many new problems and solutions for the fifth edition and who offered many suggestions for improving the consistency and clarity of the presentation. The authors especially thank the most recent reviewers of the text. Among others, they are Clark Guest, University of California, San Diego Jayantha Herath, St Cloud State University Nagarajan Kandasamy, Drexel University Avinash Karanth Kodi, Ohio University Jacob Savir, Newark College of Engineering Melissa C. Smith, Clemson University Larry M. Stephens, University of South Carolina Feedback from the readers, both critical and appreciative, is welcome. Please send your comments, concerns, and suggestions to Charles H. Roth, Jr. Larry L. Kinney How to Use This Book for Self-Study If you wish to learn all of the material in this text to mastery level, the following study procedures are recommended for each unit: 1. 2. 3. 4. 5. 6. Read the Objectives of the unit. These objectives provide a concise summary of what you should be able to do when you complete studying the unit. Work through the Study Guide. After reading each section of the text, write out the answers to the corresponding study guide questions. In many cases, blank spaces are left in the study guide so that you can write your answers directly in this book. By doing this, you will have the answers conveniently available for later review. The study guide questions generally will help emphasize some of the important points in each section or will guide you to a better understanding of some of the more difficult points. If you cannot answer some of the study guide questions, this indicates that you need to study the corresponding section in the text more before proceeding. The answers to selected study guide questions are given in the back of this book; answers to the remaining questions generally can be found within the text. Several of the units (Units 3, 5, 6, 11, 13, 14, and 18) contain one or more programmed exercises. Each programmed exercise will guide you step-by-step through the solution of one of the more difficult types of problems encountered in this text. When working through a programmed exercise, be sure to write down your answer for each part in the space provided before looking at the answer and continuing with the next part of the exercise. Work the assigned Problems at the end of the unit. Check your answers against those at the end of the book and rework any problems that you missed. Reread the Objectives of the unit to make sure that you can meet all of them. If in doubt, review the appropriate sections of the text. If you are using this text in a self-paced course, you will need to pass a readiness test on each unit before proceeding with the next unit. The purpose of the readiness test is to make sure that you have mastered the objectives of one unit before moving on to the next unit. The questions on the test will relate directly to the objectives of the unit, so that if you have worked through the study guide and written out answers to all of the study guide questions and to the problems assigned in the study guide, you should have no difficulty passing the test. xxii www.allitebooks.com About the Authors Charles H. Roth, Jr. is Professor Emeritus of Electrical and Computer Engineering at the University of Texas at Austin. He has been on the UT faculty since 1961. He received his BSEE degree from the University of Minnesota, his MSEE and EE degrees from the Massachusetts Institute of Technology, and his PhD degree in EE from Stanford University. His teaching and research interests included logic design, digital systems design, switching theory, microprocessor systems, and computeraided design. He developed a self-paced course in logic design which formed the basis of his textbook, Fundamentals of Logic Design. He is also the author of Digital Systems Design Using VHDL, two other textbooks, and several software packages. He is the author or co-author of more than 50 technical papers and reports. Six PhD students and 80 MS students have received their degrees under his supervision. He received several teaching awards including the 1974 General Dynamics Award for Outstanding Engineering Teaching. Larry L. Kinney is Professor Emeritus in Electrical and Computer Engineering at the University of Minnesota Twin Cities. He received the BS, MS, and PhD in Electrical Engineering from the University of Iowa in 1964, 1965, and 1968, respectively, and joined the University of Minnesota in 1968. He has taught a wide variety of courses including logic design, microprocessor/microcomputer systems, computer design, switching theory, communication systems and error-correcting codes. His major areas of research interest are testing of digital systems, built-in self-test, computer design, microprocessor-based systems, and error-correcting codes. xxiii UNIT Introduction Number Systems and Conversion 1 Objectives 1. Introduction The first part of this unit introduces the material to be studied later. In addition to getting an overview of the material in the first part of the course, you should be able to explain a. The difference between analog and digital systems and why digital systems are capable of greater accuracy b. The difference between combinational and sequential circuits c. Why two-valued signals and binary numbers are commonly used in digital systems 2. Number systems and conversion When you complete this unit, you should be able to solve the following types of problems: a. Given a positive integer, fraction, or mixed number in any base (2 through 16); convert to any other base. Justify the procedure used by using a power series expansion for the number. b. Add, subtract, multiply, and divide positive binary numbers. Explain the addition and subtraction process in terms of carries and borrows. c. Write negative binary numbers in sign and magnitude, 1’s complement, and 2’s complement forms. Add signed binary numbers using 1’s complement and 2’s complement arithmetic. Justify the methods used. State when an overflow occurs. d. Represent a decimal number in binary-coded-decimal (BCD), 6-3-1-1 code, excess-3 code, etc. Given a set of weights, construct a weighted code. 1 2 Unit 1 Study Guide 1. Study Section 1.1, Digital Systems and Switching Circuits, and answer the following study questions: (a) What is the basic difference between analog and digital systems? (b) Why are digital systems capable of greater accuracy than analog systems? (c) Explain the difference between combinational and sequential switching circuits. (d) What common characteristic do most switching devices used in digital systems have? (e) Why are binary numbers used in digital systems? 2. Study Section 1.2, Number Systems and Conversion. Answer the following study questions as you go along: (a) Is the first remainder obtained in the division method for base conversion the most or least significant digit? (b) Work through all of the examples in the text as you encounter them and make sure that you understand all of the steps. (c) An easy method for conversion between binary and hexadecimal is illustrated in Equation (1-1). Why should you start forming the groups of four bits at the binary point instead of the left end of the number? (d) Why is it impossible to convert a decimal number to binary on a digit-by-digit basis as can be done for hexadecimal? Number Systems and Conversion 3 (e) Complete the following conversion table. Binary (base 2) 0 1 10 11 100 101 110 111 1000 1001 1010 1011 1100 1101 1110 1111 10000 Octal (base 8) 0 Decimal (base 10) 0 Hexadecimal (base 16) 0 20 16 10 (f) Work Problems 1.1, 1.2, 1.3, and 1.4. 3. Study Section 1.3, Binary Arithmetic. (a) Make sure that you can follow all of the examples, especially the propagation of borrows in the subtraction process. (b) To make sure that you understand the borrowing process, work out a detailed analysis in terms of powers of 2 for the following example: 1100 − 101 111 4. Work Problems 1.5, 1.6, and 1.17(a). 5. Study Section 1.4, Representation of Negative Numbers. (a) In digital systems, why are 1’s complement and 2’s complement commonly used to represent negative numbers instead of sign and magnitude? 4 Unit 1 (b) State two different ways of forming the 1’s complement of an n-bit binary number. (c) State three different ways of forming the 2’s complement of an n-bit binary number. (d) If the word length is n = 4 bits (including sign), what decimal number does 10002 represent in sign and magnitude? In 2’s complement? In 1’s complement? (e) Given a negative number represented in 2’s complement, how do you find its magnitude? Given a negative number represented in 1’s complement, how do you find its magnitude? (f) If the word length is 6 bits (including sign), what decimal number does 1000002 represent in sign and magnitude? In 2’s complement? In 1’s complement? (g) What is meant by an overflow? How can you tell that an overflow has occurred when performing 1’s or 2’s complement addition? Does a carry out of the last bit position indicate that an overflow has occurred? Number Systems and Conversion 5 (h) Work out some examples of 1’s and 2’s complement addition for various combinations of positive and negative numbers. (i) What is the justification for using the end-around carry in 1’s complement addition? (j) The one thing that causes the most trouble with 2’s complement numbers is the special case of the negative number which consists of a 1 followed by all 0’s (1000 . . . 000). If this number is n bits long, what number does it represent and why? (It is not negative zero.) (k) Work Problems 1.7 and 1.8. 6. Study Section 1.5, Binary Codes. (a) Represent 187 in BCD code, excess-3 code, 6-3-1-1 code, and 2-out-of-5 code. (b) Verify that the 6-3-1-1 code is a weighted code. Note that for some decimal digits, two different code combinations could have been used. For example, either 0101 or 0110 could represent 4. In each case the combination with the smaller binary value has been used. (c) How is the excess-3 code obtained? (d) How are the ASCII codes for the decimal digits obtained? What is the relation between the ASCII codes for the capital letters and lowercase letters? (e) Work Problem 1.9. 7. If you are taking this course on a self-paced basis, you will need to pass a readiness test on this unit before going on to the next unit. The purpose of the readiness test is to determine if you have mastered the material in this unit and are ready to go on to the next unit. Before you take the readiness test: (a) Check your answers to the problems against those provided at the end of this book. If you missed any of the problems, make sure that you understand why your answer is wrong and correct your solution. (b) Make sure that you can meet all of the objectives listed at the beginning of this unit. Introduction Number Systems and Conversion 1.1 Digital Systems and Switching Circuits Digital systems are used extensively in computation and data processing, control systems, communications, and measurement. Because digital systems are capable of greater accuracy and reliability than analog systems, many tasks formerly done by analog systems are now being performed digitally. In a digital system, the physical quantities or signals can assume only discrete values, while in analog systems the physical quantities or signals may vary continuously over a specified range. For example, the output voltage of a digital system might be constrained to take on only two values such as 0 volts and 5 volts, while the output voltage from an analog system might be allowed to assume any value in the range −10 volts to +10 volts. Because digital systems work with discrete quantities, in many cases they can be designed so that for a given input, the output is exactly correct. For example, if we multiply two 5-digit numbers using a digital multiplier, the 10-digit product will be correct in all 10 digits. On the other hand, the output of an analog multiplier might have an error ranging from a fraction of one percent to a few percent depending on the accuracy of the components used in construction of the multiplier. Furthermore, if we need a product which is correct to 20 digits rather than 10, we can redesign the digital multiplier to process more digits and add more digits to its input. A similar improvement in the accuracy of an analog multiplier would not be possible because of limitations on the accuracy of the components. The design of digital systems may be divided roughly into three parts—system design, logic design, and circuit design. System design involves breaking the overall system into subsystems and specifying the characteristics of each subsystem. For example, the system design of a digital computer could involve specifying the number and type of memory units, arithmetic units, and input-output devices as well as the interconnection and control of these subsystems. Logic design involves determining how to interconnect basic logic building blocks to perform a specific function. An example of logic design is determining the interconnection of logic gates and flip-flops required to perform binary addition. Circuit design involves specifying the interconnection of specific components such as resistors, diodes, and transistors 6 Number Systems and Conversion 7 FIGURE 1-1 Switching Circuit Inputs Xm ... © Cengage Learning 2014 X1 X2 Switching Circuit ... to form a gate, flip-flop, or other logic building block. Most contemporary circuit design is done in integrated circuit form using appropriate computer-aided design tools to lay out and interconnect the components on a chip of silicon. This book is largely devoted to a study of logic design and the theory necessary for understanding the logic design process. Some aspects of system design are treated in Units 18 and 20. Circuit design of logic gates is discussed briefly in Appendix A. Many of a digital system’s subsystems take the form of a switching circuit (Figure 1-1). A switching circuit has one or more inputs and one or more outputs which take on discrete values. In this text, we will study two types of switching circuits—combinational and sequential. In a combinational circuit, the output values depend only on the present value of the inputs and not on past values. In a sequential circuit, the outputs depend on both the present and past input values. In other words, in order to determine the output of a sequential circuit, a sequence of input values must be specified. The sequential circuit is said to have memory because it must “remember” something about the past sequence of inputs, while a combinational circuit has no memory. In general, a sequential circuit is composed of a combinational circuit with added memory elements. Combinational circuits are easier to design than sequential circuits and will be studied first. Z1 Z2 Outputs Zn The basic building blocks used to construct combinational circuits are logic gates. The logic designer must determine how to interconnect these gates in order to convert the circuit input signals into the desired output signals. The relationship between these input and output signals can be described mathematically using Boolean algebra. Units 2 and 3 of this text introduce the basic laws and theorems of Boolean algebra and show how they can be used to describe the behavior of circuits of logic gates. Starting from a given problem statement, the first step in designing a combinational logic circuit is to derive a table or the algebraic logic equations which describe the circuit outputs as a function of the circuit inputs (Unit 4). In order to design an economical circuit to realize these output functions, the logic equations which describe the circuit outputs generally must be simplified. Algebraic methods for this simplification are described in Unit 3, and other simplification methods (Karnaugh map and Quine-McCluskey procedure) are introduced in Units 5 and 6. Implementation of the simplified logic equations using several types of gates is described in Unit 7, and alternative design procedures using programmable logic devices are developed in Unit 9. The basic memory elements used in the design of sequential circuits are called flip-flops (Unit 11). These flip-flops can be interconnected with gates to form counters and registers (Unit 12). Analysis of more general sequential circuits using timing 8 Unit 1 diagrams, state tables, and graphs is presented in Unit 13. The first step in designing a sequential switching circuit is to construct a state table or graph which describes the relationship between the input and output sequences (Unit 14). Methods for going from a state table or graph to a circuit of gates and flip-flops are developed in Unit 15. Methods of implementing sequential circuits using programmable logic are discussed in Unit 16. In Unit 18, combinational and sequential design techniques are applied to the realization of systems for performing binary addition, multiplication, and division. The sequential circuits designed in this text are called synchronous sequential circuits because they use a common timing signal, called a clock, to synchronize the operation of the memory elements. Use of a hardware description language, VHDL, in the design of combinational logic, sequential logic, and digital systems is introduced in Units 10, 17, and 20. VHDL is used to describe, simulate, and synthesize digital hardware. After writing VHDL code, the designer can use computer-aided design software to compile the hardware description and complete the design of the digital logic. This allows the completion of complex designs without having to manually work out detailed circuit descriptions in terms of gates and flip-flops. The switching devices used in digital systems are generally two-state devices, that is, the output can assume only two different discrete values. Examples of switching devices are relays, diodes, and transistors. A relay can assume two states—closed or open—depending on whether power is applied to the coil or not. A diode can be in a conducting state or a nonconducting state. A transistor can be in a cut-off or saturated state with a corresponding high or low output voltage. Of course, transistors can also be operated as linear amplifiers with a continuous range of output voltages, but in digital applications greater reliability is obtained by operating them as twostate devices. Because the outputs of most switching devices assume only two different values, it is natural to use binary numbers internally in digital systems. For this reason binary numbers and number systems will be discussed first before proceeding to the design of switching circuits. 1.2 Number Systems and Conversion When we write decimal (base 10) numbers, we use a positional notation; each digit is multiplied by an appropriate power of 10 depending on its position in the number. For example, 953.7810 = 9 × 102 + 5 × 101 + 3 × 100 + 7 × 10 −1 + 8 × 10 −2 Similarly, for binary (base 2) numbers, each binary digit is multiplied by the appropriate power of 2: 1011.112 = 1 × 23 + 0 × 22 + 1 × 21 + 1 × 20 + 1 × 2 −1 + 1 × 2 −2 = 8 + 0 + 2 + 1 + 12 + 14 = 1134 = 11.7510 www.allitebooks.com Number Systems and Conversion 9 Note that the binary point separates the positive and negative powers of 2 just as the decimal point separates the positive and negative powers of 10 for decimal numbers. Any positive integer R (R > 1) can be chosen as the radix or base of a number system. If the base is R, then R digits (0, 1, . . . , R − 1) are used. For example, if R = 8, then the required digits are 0, 1, 2, 3, 4, 5, 6, and 7. A number written in positional notation can be expanded in a power series in R. For example, N = (a4 a3 a2 a1a0 . a −1a −2 a −3)R = a4 × R4 + a3 × R3 + a2 × R2 + a1 × R1 + a0 × R0 + a −1 × R −1 + a −2 × R −2 + a −3 × R −3 where ai is the coefficient of Ri and 0 ≤ ai ≤ R − 1. If the arithmetic indicated in the power series expansion is done in base 10, then the result is the decimal equivalent of N. For example, 147.38 = 1 × 82 + 4 × 81 + 7 × 80 + 3 × 8 −1 = 64 + 32 + 7 + = 103.37510 3 8 The power series expansion can be used to convert to any base. For example, converting 14710 to base 3 would be written as 14710 = 1 × (101)2 + (11) × (101)1 + (21) × (101)0 where all the numbers on the right-hand side are base 3 numbers. (Note: In base 3, 10 is 101, 7 is 21, etc.) To complete the conversion, base 3 arithmetic would be used. Of course, this is not very convenient if the arithmetic is being done by hand. Similarly, if 14710 is being converted to binary, the calculation would be 14710 = 1 × (1010)2 + (100) × (1010)1 + (111) × (1010)0 Again this is not convenient for hand calculation but it could be done easily in a computer where the arithmetic is done in binary. For hand calculation, use the power series expansion when converting from some base into base 10. For bases greater than 10, more than 10 symbols are needed to represent the digits. In this case, letters are usually used to represent digits greater than 9. For example, in hexadecimal (base 16), A represents 1010, B represents 1110, C represents 1210, D represents 1310, E represents 1410, and F represents 1510. Thus, A2F16 = 10 × 162 + 2 × 161 + 15 × 160 = 2560 + 32 + 15 = 260710 Next, we will discuss conversion of a decimal integer to base R using the division method. The base R equivalent of a decimal integer N can be represented as N = (an an−1 · · · a2 a1 a0)R = an Rn + an−1Rn−1 + · · · + a2R2 + a1R1 + a0 10 Unit 1 If we divide N by R, the remainder is a0: N = an Rn−1 + an−1Rn−2 + · · · + a2 R1 + a1 = Q1, remainder a0 R Then we divide the quotient Q1 by R: Q1 = an Rn−2 + an−1Rn−3 + · · · + a3 R1 + a2 = Q2, remainder a1 R Next we divide Q2 by R: Q2 = an Rn−3 + an−1Rn−4 + · · · + a3 = Q3, remainder a2 R This process is continued until we finally obtain an. Note that the remainder obtained at each division step is one of the desired digits and the least significant digit is obtained first. Example Convert 5310 to binary. 2 y53 2 y26 2 y13 2 y6 2 y3 2 y1 0 rem. rem. rem. rem. rem. rem. = = = = = = 1 = a0 0 = a1 1 = a2 0 = a3 1 = a4 1 = a5 5310 = 1101012 Conversion of a decimal fraction to base R can be done using successive multiplications by R. A decimal fraction F can be represented as F = (.a −1 a −2 a −3 · · · a −m)R = a −1R −1 + a −2 R −2 + a −3 R −3 + · · · + a −m R −m Multiplying by R yields FR = a −1 + a −2 R −1 + a −3 R −2 + · · · + a −m R −m+1 = a −1 + F1 where F1 represents the fractional part of the result and a −1 is the integer part. Multiplying F1 by R yields F1R = a −2 + a −3 R −1 + · · · + a −m R −m+2 = a −2 + F2 Number Systems and Conversion 11 Next, we multiply F2 by R: F2R = a −3 + · · · + a −m R −m+3 = a −3 + F3 This process is continued until we have obtained a sufficient number of digits. Note that the integer part obtained at each step is one of the desired digits and the most significant digit is obtained first. Example Convert 0.62510 to binary. F = .625 × 2 1.250 (a−1 = 1) F1 = .250 × 2 0.500 (a−2 = 0) F2 = .500 × 2 1.000 (a−3 = 1) .62510 = .1012 This process does not always terminate, but if it does not terminate, the result is a repeating fraction. Example Convert 0.710 to binary. .7 2 (1).4 2 (0).8 2 (1).6 2 (1).2 2 (0).4 2 (0).8 ⟵ process starts repeating here because 0.4 was previously obtained 0.710 = 0.1 0110 0110 0110 . . . 2 Conversion between two bases other than decimal can be done directly by using the procedures given; however, the arithmetic operations would have to be carried out using a base other than 10. It is generally easier to convert to decimal first and then convert the decimal number to the new base. 12 Unit 1 Example Convert 231.34 to base 7. 231.34 = 2 × 16 + 3 × 4 + 1 + 34 = 45.7510 7 y45 7y 6 0 rem. 3 rem. 6 .75 7 (5) .25 7 (1) .75 7 (5) .25 7 (1) .75 45.7510 = 63.5151 . . . 7 Conversion from binary to hexadecimal (and conversely) can be done by inspection because each hexadecimal digit corresponds to exactly four binary digits (bits). Starting at the binary point, the bits are divided into groups of four, and each group is replaced by a hexadecimal digit: ('* 1101 ('* . 0101 ('* 1100 ('* = 4D.5C16 1001101.0101112 = 0100 4 D 5 C (1-1) As shown in Equation (1-1), extra 0’s are added at each end of the bit string as needed to fill out the groups of four bits. 1.3 Binary Arithmetic Arithmetic operations in digital systems are usually done in binary because design of logic circuits to perform binary arithmetic is much easier than for decimal. Binary arithmetic is carried out in much the same manner as decimal, except the addition and multiplication tables are much simpler. The addition table for binary numbers is 0+0=0 0+1=1 1+0=1 1+1=0 and carry 1 to the next column Carrying 1 to a column is equivalent to adding 1 to that column. Number Systems and Conversion Example 13 Add 1310 and 1110 in binary. 1 1 1 1 ⟵ carries 1310 = 1101 1110 = 1011 11000 = 2410 The subtraction table for binary numbers is 0−0=0 0−1=1 1−0=1 1−1=0 and borrow 1 from the next column Borrowing 1 from a column is equivalent to subtracting 1 from that column. Examples of Binary Subtraction (a) 1← (indicates 11101 a borrrow – 10011 from the 1010 3rd column) (b) 1 1 1 1← borrows 10000 – 11 1101 (c) 1 1 1← borrows 111001 – 1011 101110 Note how the borrow propagates from column to column in the second example. In order to borrow 1 from the second column, we must in turn borrow 1 from the third column, etc. An alternative to binary subtraction is the use of 2’s complement arithmetic, as discussed in Section 1.4. Binary subtraction sometimes causes confusion, perhaps because we are so used to doing decimal subtraction that we forget the significance of the borrowing process. Before doing a detailed analysis of binary subtraction, we will review the borrowing process for decimal subtraction. If we number the columns (digits) of a decimal integer from right to left (starting with 0), and then we borrow 1 from column n, what we mean is that we subtract 1 from column n and add 10 to column n − 1. Because 1 × 10n = 10 × 10n−1, the value of the decimal number is unchanged, but we can proceed with the subtraction. Consider, for example, the following decimal subtraction problem: column 2⟶ column 1 ↙ 205 − 18 187 14 Unit 1 A detailed analysis of the borrowing process for this example, indicating first a borrow of 1 from column 1 and then a borrow of 1 from column 2, is as follows: 205 − 18 = [2 × 102 + 0 × 101 + 5 × 100] 1 × 101 + 8 × 100] −[note borrow from column 1 = [2 × 102 + (0 − 1) × 101 + (10 + 5) × 100] 1 × 101 + 8 × 100] −[note borrow from column 2 = − = [(2 − 1) × 102 + (10 + 0 − 1) × 101 + 15 × 100] [1 × 101 + 8 × 100] [1 × 102 + 8 × 101 + 7 × 100] = 187 The analysis of borrowing for binary subtraction is exactly the same, except that we work with powers of 2 instead of powers of 10. Thus for a binary number, borrowing 1 from column n is equivalent to subtracting 1 from column n and adding 2 (102) to column n − 1. The value of the binary number is unchanged because 1 × 2n = 2 × 2n−1. A detailed analysis of binary subtraction example (c) follows. Starting with the rightmost column, 1 − 1 = 0. To subtract in the second column, we must borrow from the third column. Rather than borrow immediately, we place a 1 over the third column to indicate that a borrow is necessary, and we will actually do the borrowing when we get to the third column. (This is similar to the way borrow signals might propagate in a computer.) Now because we have borrowed 1, the second column becomes 10, and 10 − 1 = 1. In order to borrow 1 from the third column, we must borrow 1 from the fourth column (indicated by placing a 1 over column 4). Column 3 then becomes 10, subtracting off the borrow yields 1, and 1 − 0 = 1. Now in column 4, we subtract off the borrow leaving 0. In order to complete the subtraction, we must borrow from column 5, which gives 10 in column 4, and 10 − 1 = 1. The multiplication table for binary numbers is 0× 0× 1× 1× 0= 1= 0= 1= 0 0 0 1 The following example illustrates multiplication of 1310 by 1110 in binary: 1101 1011 1101 1101 0000 1101 10001111 = 14310 Number Systems and Conversion 15 Note that each partial product is either the multiplicand (1101) shifted over the appropriate number of places or is zero. When adding up long columns of binary numbers, the sum of the bits in a single column can exceed 112, and therefore the carry to the next column can be greater than 1. For example, if a single column of bits contains five 1’s, then adding up the 1’s gives 1012, which means that the sum bit for that column is 1, and the carry to the next column is 102. When doing binary multiplication, a common way to avoid carries greater than 1 is to add in the partial products one at a time as illustrated by the following example: 1111 1101 1111 0000 (01111) 1111 (1001011) 1111 11000011 multiplicand multiplier first partial product second partial product sum of first two partial products third partial product sum after adding third partial product fourth partial product final product (sum after adding fourth partial product) The following example illustrates division of 14510 by 1110 in binary: 1101 1011 1001001 1011 1110 1011 1101 1011 10 | The quotient is 1101 with a remainder of 10. Binary division is similar to decimal division, except it is much easier because the only two possible quotient digits are 0 and 1. In the above example, if we start by comparing the divisor (1011) with the upper four bits of the dividend (1001), we find that we cannot subtract without a negative result, so we move the divisor one place to the right and try again. This time we can subtract 1011 from 10010 to give 111 as a result, so we put the first quotient bit of 1 above 10010. We then bring down the next dividend bit (0) to get 1110 and shift the divisor right. We then subtract 1011 from 1110 to get 11, so the second quotient bit is 1. When we bring down the next dividend bit, the result is 110, and we cannot subtract the shifted divisor, so the third quotient bit is 0. We then bring down the last dividend bit and subtract 1011 from 1101 to get a final remainder of 10, and the last quotient bit is 1. 16 Unit 1 1.4 Representation of Negative Numbers Up to this point we have been working with unsigned positive numbers. The most common methods for representing both positive and negative numbers are sign and magnitude, 2’s complement, and 1’s complement. In each of these methods, the leftmost bit of a number is 0 for positive numbers and 1 for negative numbers. As discussed below, if n bits are used to represent numbers, then the sign and magnitude and 1’s complement methods represent numbers in the range −(2(n−1) − 1) to +(2(n−1) − 1) and both have two representations for 0, a positive 0 and a negative 0. In 2’s complement, numbers in the range −2(n−1) to +(2(n−1) − 1) are represented and there is only a positive 0. If an operation, such as addition or subtraction, is performed on two numbers and the result is outside the range of representation, then we say that an overflow has occurred. Sign and Magnitude Numbers In an n-bit sign and magnitude system, a number is represented by a sign bit, 0 for positive and 1 for negative, followed by n − 1 bits that represent the magnitude of the number. With n − 1 bits the magnitude can be 0 to 2(n−1) − 1. With the sign bit, numbers in the range −(2(n−1) − 1) to +(2(n−1) − 1) are represented including a positive and negative 0. This is illustrated in Table 1-1 for n = 4. For example, 0011 represents +3 and 1011 represents −3. Note that 1000 represents minus 0. Designing logic circuits to perform arithmetic on sign and magnitude binary numbers is awkward. One method is to convert the numbers into 2’s (or 1’s) complement and, after performing the arithmetic operation, convert the result back to sign and magnitude. TABLE 1-1 Signed Binary Integers (word length: n = 4) © Cengage Learning 2014 +N +0 +1 +2 +3 +4 +5 +6 +7 Positive Integers (all systems) 0000 0001 0010 0011 0100 0101 0110 0111 Negative Integers −N −0 −1 −2 −3 −4 −5 −6 −7 −8 Sign and Magnitude 1000 1001 1010 1011 1100 1101 1110 1111 —— 2’s Complement N* —— 1111 1110 1101 1100 1011 1010 1001 1000 1’s Complement N 1111 1110 1101 1100 1011 1010 1001 1000 —— 2’s Complement Numbers In the 2’s complement number system, a positive number, N, is represented by a 0 followed by the magnitude of N as in the sign and magnitude system; however, Number Systems and Conversion 17 a negative number, −N, is represented by its 2’s complement, N*. If the word length is n bits, the 2’s complement of a positive integer N is defined as N* = 2n − N (1-2) (Note that in this equation all numbers N, 2n, and N* are treated as unsigned positive numbers; if they are expressed in binary, n + 1 bits are required to represent 2n.) Table 1-1 shows the result for n = 4. In Table 1-1, the 2’s complement representation of negative numbers −1 through −7 can be obtained by taking the 2’s complement of positive numbers 1 through 7 (i.e., by subtracting from 16). For example, the 2’s complement of 5 is 16 − 5 = 11 or, using binary numbers, (10000) − (0101) = (1011). After completing the subtractions, all combinations of 4-bits have been used to represent the numbers −7, . . . , −1, 0, 1, . . . 7; the only unused combination is 1000. Since the leftmost bit of 1000 is 1, it should be a negative number. To determine its magnitude, note that the magnitude of a negative number can be obtained by taking its 2’s complement; that is, from Equation (1-2), N = 2n − N* (1-3) Applying Equation (1-3) to 1000 produces (10000) − (1000) = (1000) or, in decimal, 16 − 8 = 8. Hence, 1000 represents −8. In general, in an n-bit 2’s complement system the number 1 followed by all 0’s represents −2(n−1). Using Equation (1-2) directly on binary numbers requires subtraction of n + 1 bit numbers. This can be avoided by noting that Equation (1-2) can be written as N* = (2n − 1 − N) + 1 In binary, 2n − 1 consists of n 1’s. Subtracting a number from all 1’s does not produce any borrows, and the subtraction can be done by replacing 0’s with 1’s and 1’s with 0’s (i.e., simply complement N bit-by-bit). For example, if n = 7 and N = 0101100, 2n − 1 = 1111111 − 0101100 1010011 + 0000001 N* = 1010100 N* is obtained by complementing N bit-by-bit and then adding 1. An alternative way to form the 2’s complement of N is to start at the right and leave any 0’s on the right end and the first 1 unchanged, then complement all bits to the left of the first 1. In the preceding example, the 100 on the right end of N is unchanged while the 0101 on the left is complemented bit-by-bit. Addition of 2’s Complement Numbers The addition of n-bit signed binary numbers is straightforward using the 2’s complement system. The addition is carried out just as if all the numbers were positive, and any carry from the sign position is ignored. This will always yield the correct result 18 Unit 1 except when an overflow occurs. When the word length is n bits, we say that an overflow has occurred if the correct representation of the sum (including sign) requires more than n bits. The different cases which can occur are illustrated below for n = 4. 1. Addition of two positive numbers, sum < 2n−1 +3 +4 +7 2. 0101 1010 1111 (correct answer) 1011 0110 (1)0001 ⟵ correct answer when the carry from the sign bit is ignored (this is not an overflow) Addition of two negative numbers, 0 sum 0 ≤ 2n−1 −3 −4 −7 6. ⟵ wrong answer because of overflow (+11 requires 5 bits including sign) Same as case 3 except positive number has greater magnitude +5 +6 +1 5. 0101 0110 1011 Addition of positive and negative numbers (negative number has greater magnitude) −5 −6 −1 4. (correct answer) Addition of two positive numbers, sum ≥ 2n−1 +5 +6 3. 0011 0100 0111 1101 1100 (1)1001 ⟵ correct answer when the last carry is ignored (this is not an overflow) Addition of two negative numbers, 0 sum 0 > 2n−1 −5 −6 1011 1010 (1)0101 ⟵ wrong answer because of overflow (−11 requires 5 bits including sign) Note that an overflow condition (cases 2 and 6) is easy to detect because in case 2 the addition of two positive numbers yields a negative result, and in case 6 the addition of two negative numbers yields a positive answer (for four bits). www.allitebooks.com Number Systems and Conversion 19 The proof that throwing away the carry from the sign bit always gives the correct answer follows for cases 4 and 5: Case 4: −A + B (where B > A) A* + B = (2n − A) + B = 2n + (B − A) > 2n Throwing away the last carry is equivalent to subtracting 2n, so the result is (B − A), which is correct. Case 5: −A − B (where A + B ≤ 2n−1) A* + B* = (2n − A) + (2n − B) = 2n + 2n − (A + B) Discarding the last carry yields 2n − (A + B) = (A + B)*, which is the correct representation of −(A + B). 1’s Complement Numbers In the 1’s complement system a negative number, −N, is represented by the 1’s complement of N, N, defined as N = (2n − 1) − N (1-4) As explained above, (2n − 1) consists of all 1’s, and subtracting a bit from 1 is the same as complementing the bit. Hence, the 1’s complement of N can be obtained by complementing N bit-by-bit. Table 1-1 illustrates 1’s complement for n = 4. Note that the 1’s complement of 0000 is 1111, which represents minus zero. Note that 1’s complement has two representations of 0, as does sign and magnitude. Addition of 1’s Complement Numbers The addition of 1’s complement numbers is similar to 2’s complement except that instead of discarding the last carry, it is added to the n-bit sum in the position furthest to the right. This is referred to as an end-around carry. The addition of positive numbers is the same as illustrated for cases 1 and 2 under 2’s complement. The remaining cases are illustrated below (n = 4). 3. Addition of positive and negative numbers (negative number with greater magnitude) +5 −6 −1 4. 0101 1001 1110 (correct answer) Same as case 3 except positive number has greater magnitude −5 +6 1010 0110 (1) 0000 1 0001 (end-around carry) (correct answer, no overflow) 20 Unit 1 5. Addition of two negative numbers, 0 sum 0 < 2n−1 −3 −4 6. 1100 1011 (1) 0111 1 1000 (end-around carry) (correct answer, no overflow) Addition of two negative numbers, |sum| ≥ 2n−1 −5 −6 1010 1001 (1) 0111 1 0100 (end-around carry) (wrong answer because of overflow) Again, note that the overflow in case 6 is easy to detect because the addition of two negative numbers yields a positive result. The proof that the end-around carry method gives the correct result follows for cases 4 and 5: Case 4: −A + B (where B > A) A + B = (2n − 1 − A) + B = 2n + (B − A) − 1 The end-around carry is equivalent to subtracting 2n and adding 1, so the result is (B − A), which is correct. Case 5: −A − B (A + B < 2n−1) A + B = (2n − 1 − A) + (2n − 1 − B) = 2n + [2n − 1 − (A + B)] − 1 After the end-around carry, the result is 2n − 1 − (A + B) = (A + B) which is the correct representation for −(A + B). The following examples illustrate the addition of 1’s and 2’s complement numbers for a word length of n = 8: 1. Add −11 and −20 in 1’s complement. +11 = 00001011 +20 = 00010100 taking the bit-by-bit complement, −11 is represented by 11110100 and −20 by 11101011 11110100 (−11) 11101011 +(−20) (1) 11011111 1 (end-around carry) 11100000 = −31 Number Systems and Conversion 2. 21 Add −8 and +19 in 2’s complement +8 = 00001000 complementing all bits to the left of the first 1, −8, is represented by 11111000 11111000 (−8) 00010011 +19 (1)00001011 = +11 (discard last carry) Note that in both cases, the addition produced a carry out of the furthest left bit position, but there is no overflow because the answer can be correctly represented by eight bits (including sign). A general rule for detecting overflow when adding two n-bit signed binary numbers (1’s or 2’s complement) to get an n-bit sum is: An overflow occurs if adding two positive numbers gives a negative answer or if adding two negative numbers gives a positive answer. An alternative method for detecting overflow in 2’s complement addition is as follows: An overflow occurs if and only if the carry out of the sign position is not equal to the carry into the sign position. 1.5 Binary Codes Although most large computers work internally with binary numbers, the inputoutput equipment generally uses decimal numbers. Because most logic circuits only accept two-valued signals, the decimal numbers must be coded in terms of binary signals. In the simplest form of binary code, each decimal digit is replaced by its binary equivalent. For example, 937.25 is represented by 9 3 7 . 2 5 C 0011 C 0111 C . 0010 C 0101 C 1001 This representation is referred to as binary-coded-decimal (BCD) or more explicitly as 8-4-2-1 BCD. Note that the result is quite different than that obtained by converting the number as a whole into binary. Because there are only ten decimal digits, 1010 through 1111 are not valid BCD codes. 22 Unit 1 TABLE 1-2 Binary Codes for Decimal Digits © Cengage Learning 2014 Decimal Digit 8-4-2-1 Code (BCD) 6-3-1-1 Code Excess-3 Code 2-out-of-5 Code Gray Code 0 1 2 3 4 5 6 7 8 9 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 0000 0001 0011 0100 0101 0111 1000 1001 1011 1100 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 00011 00101 00110 01001 01010 01100 10001 10010 10100 11000 0000 0001 0011 0010 0110 1110 1010 1011 1001 1000 Table 1-2 shows several possible sets of binary codes for the ten decimal digits. Many other possibilities exist because the only requirement for a valid code is that each decimal digit be represented by a distinct combination of binary digits. To translate a decimal number to coded form, each decimal digit is replaced by its corresponding code. Thus 937 expressed in excess-3 code is 1100 0110 1010. The 8-4-2-1 (BCD) code and the 6-3-1-1 code are examples of weighted codes. A 4-bit weighted code has the property that if the weights are w3, w2, w1, and w0, the code a3a2a1a0 represents a decimal number N, where N = w3 a3 + w2 a2 + w1a1 + w0 a0 For example, the weights for the 6-3-1-1 code are w3 = 6, w2 = 3, w1 = 1, and w0 = 1. The binary code 1011 thus represents the decimal digit N = 6·1 + 3·0 + 1·1 + 1·1 = 8 The excess-3 code is obtained from the 8-4-2-1 code by adding 3 (0011) to each of the codes. The 2-out-of-5 code has the property that exactly 2 out of the 5 bits are 1 for every valid code combination. This code has useful error-checking properties because if any one of the bits in a code combination is changed due to a malfunction of the logic circuitry, the number of 1 bits is no longer exactly two. The table shows one example of a Gray code. A Gray code has the property that the codes for successive decimal digits differ in exactly one bit. For example, the codes for 6 and 7 differ only in the fourth bit, and the codes for 9 and 0 differ only in the first bit. A Gray code is often used when translating an analog quantity, such as a shaft position, into digital form. In this case, a small change in the analog quantity will change only one bit in the code, which gives more reliable operation than if two or more bits changed at a time. The Gray and 2-out-of-5 codes are not weighted codes. In general, the decimal value of a coded digit cannot be computed by a simple formula when a non-weighted code is used. Many applications of computers require the processing of data which contains numbers, letters, and other symbols such as punctuation marks. In order to transmit Number Systems and Conversion 23 such alphanumeric data to or from a computer or store it internally in a computer, each symbol must be represented by a binary code. One common alphanumeric code is the ASCII code (American Standard Code for Information Interchange). This is a 7-bit code, so 27(128) different code combinations are available to represent letters, numbers, and other symbols. Table 1-3 shows a portion of the ASCII code; the code combinations not listed are used for special control functions such as “form feed” or “end of transmission.” The word “Start” is represented in ASCII code as follows: 1010011 1110100 1100001 1110010 1110100 S t a r t ASCII Code ASCII Code Character space ! “ # $ % & ′ ( ) * + , − . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? ASCII Code ASCII Code A6 A5 A4 A3 A2 A1 A0 Character A6 A5 A4 A3 A2 A1 A0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 1 0 1 0 0 1 1 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1 0 0 1 0 1 1 0 0 1 0 1 1 1 0 1 0 1 1 1 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 1 0 1 1 0 0 1 0 1 1 0 1 0 0 1 1 0 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 1 0 1 1 1 0 1 0 1 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [\ ] ^ — 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 Character A6 A5 A4 A3 A2 A1 A0 ' a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~ delete 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 © Cengage Learning 2014 TABLE 1-3 24 Unit 1 Problems 1.1 Convert to hexadecimal and then to binary: (a) 757.2510 (b) 123.1710 (c) 356.8910 (d) 1063.510 1.2 Convert to octal. Convert to hexadecimal. Then convert both of your answers to decimal, and verify that they are the same. (a) 111010110001.0112 (b) 10110011101.112 1.3 Convert to base 6: 3BA.2514 (do all of the arithmetic in decimal). 1.4 (a) Convert to hexadecimal: 1457.1110. Round to two digits past the hexadecimal point. (b) Convert your answer to binary, and then to octal. (c) Devise a scheme for converting hexadecimal directly to base 4 and convert your answer to base 4. (d) Convert to decimal: DEC.A16. 1.5 Add, subtract, and multiply in binary: (a) 1111 and 1010 (b) 110110 and 11101 (c) 100100 and 10110 1.6 Subtract in binary. Place a 1 over each column from which it was necessary to borrow. (a) 11110100 − 1000111 (b) 1110110 − 111101 (c) 10110010 − 111101 1.7 Add the following numbers in binary using 2’s complement to represent negative numbers. Use a word length of 6 bits (including sign) and indicate if an overflow occurs. (a) 21 + 11 (b) (−14) + (−32) (c) (−25) + 18 (d) (−12) + 13 (e) (−11) + (−21) Repeat (a), (c), (d), and (e) using 1’s complement to represent negative numbers. 1.8 A computer has a word length of 8 bits (including sign). If 2’s complement is used to represent negative numbers, what range of integers can be stored in the computer? If 1’s complement is used? (Express your answers in decimal.) 1.9 Construct a table for 7-3-2-1 weighted code and write 3659 using this code. 1.10 Convert to hexadecimal and then to binary. (a) 1305.37510 (b) 111.3310 (c) 301.1210 (d) 1644.87510 1.11 Convert to octal. Convert to hexadecimal. Then convert both of your answers to decimal, and verify that they are the same. (a) 101111010100.1012 (b) 100001101111.012 Number Systems and Conversion 25 1.12 (a) Convert to base 3: 375.548 (do all of the arithmetic in decimal). (b) Convert to base 4: 384.7410. (c) Convert to base 9: A52.A411 (do all of the arithmetic in decimal). 1.13 Convert to hexadecimal and then to binary: 544.19. 1.14 Convert the decimal number 97.710 into a number with exactly the same value represented in the following bases. The exact value requires an infinite repeating part in the fractional part of the number. Show the steps of your derivation. (a) binary (b) octal (c) hexadecimal (d) base 3 (e) base 5 1.15 Devise a scheme for converting base 3 numbers directly to base 9. Use your method to convert the following number to base 9: 1110212.202113 1.16 Convert the following decimal numbers to octal and then to binary: (a) 298363/64 (b) 93.70 (c) 298331/32 (d) 109.30 1.17 Add, subtract, and multiply in binary: (a) 1111 and 1001 (b) 1101001 and 110110 (c) 110010 and 11101 1.18 Subtract in binary. Place a 1 over each column from which it was necessary to borrow. (a) 10100100 − 01110011 (b) 10010011 − 01011001 (c) 11110011 − 10011110 1.19 Divide in binary: (a) 11101001 ÷ 101 (b) 110000001 ÷ 1110 (c) 1110010 ÷ 1001 Check your answers by multiplying out in binary and adding the remainder. 1.20 Divide in binary: (a) 10001101 ÷ 110 (b) 110000011 ÷ 1011 (c) 1110100 ÷ 1010 1.21 Assume three digits are used to represent positive integers and also assume the following operations are correct. Determine the base of the numbers. Did any of the additions overflow? (a) 654 + 013 = 000 (b) 024 + 043 + 013 + 033 = 223 (c) 024 + 043 + 013 + 033 = 201 1.22 What is the lowest number of bits (digits) required in the binary number approximately equal to the decimal number 0.611710 so that the binary number has the same or better precision? 1.23 Convert 0.363636 . . . 10 to its exact equivalent base 8 number. 26 Unit 1 1.24 (a) Verify that a number in base b can be converted to base b3 by partitioning the digits of the base b number into groups of three consecutive digits starting at the radix point and proceeding both left and right and converting each group into a base b3 digit. (Hint: Represent the base b number using the power series expansion.) (b) Verify that a number in base b3 can be converted to base b by expanding each digit of the base b3 number into three consecutive digits starting at the radix point and proceeding both left and right. 1.25 (a) Show how to represent each of the numbers (5 − 1), (52 − 1), and (53 − 1) as base 5 numbers. (b) Generalize your answers to part (a) and show how to represent (bn − 1) as a base b number, where b can be any integer larger than 1 and n any integer larger than 0. Give a mathematical derivation of your result. 1.26 (a) Show that the number 121b, where b is any base greater than 2, is a perfect square (i.e., it is equal to the square of some number). (b) Repeat part (a) for the number 12321b, where b > 3. (c) Repeat part (a) for the number 14641b, where b > 6. (d) Repeat part (a) for the number 1234321b, where b > 4. 1.27 (a) Convert (0.12)3 to a base 6 fraction. (b) Convert (0.375)10 to a base 8 fraction. (c) Let N = (0.a −1a −2 · · · a −m)R be an any base R fraction with at most m nonzero digits. Determine the necessary and sufficient conditions for N to be representable as a base S fraction with a finite number of nonzero digits; say N = (0.b −1b −2 · · · b −n)S. (Hint: Part (a) gives an example. Note that (a −1R −1 + a −2 R −2 + · · · a −m R −m)Sn must be an integer.) (d) Generalize part (a) to determine necessary and sufficient conditions for a specific, but not every, base R fraction, N = (0.a −1a −2 · · · a −m)R, to be representable as a base S fraction with a finite number of nonzero digits. 1.28 Construct a table for 4-3-2-1 weighted code and write 9154 using this code. 1.29 Is it possible to construct a 5-3-1-1 weighted code? A 6-4-1-1 weighted code? Justify your answers. 1.30 Is it possible to construct a 5-4-1-1 weighted code? A 6-3-2-1 weighted code? Justify your answers. 1.31 Construct a 6-2-2-1 weighted code for decimal digits. What number does 1100 0011 represent in this code? Number Systems and Conversion 27 1.32 Construct a 5-2-2-1 weighted code for decimal digits. What numbers does 1110 0110 represent in this code? 1.33 Construct a 7-3-2-1 code for base 12 digits. Write B4A9 using this code. 1.34 (a) It is possible to have negative weights in a weighted code for the decimal digits, e.g., 8, 4, −2, and −1 can be used. Construct a table for this weighted code. (b) If d is a decimal digit in this code, how can the code for 9 − d be obtained? 1.35 Convert to hexadecimal, and then give the ASCII code for the resulting hexadecimal number (including the code for the hexadecimal point): (b) 183.8110 (a) 222.2210 1.36 Repeat 1.7 for the following numbers: (a) (−10) + (−11) (b) (−10) + (−6) (d) 11 + 9 (e) (−11) + (−4) (c) (−8) + (−11) 1.37 Because A − B = A + (−B), the subtraction of signed numbers can be accomplished by adding the complement. Subtract each of the following pairs of 5-bit binary numbers by adding the complement of the subtrahend to the minuend. Indicate when an overflow occurs. Assume that negative numbers are represented in 1’s complement. Then repeat using 2’s complement. (a) 01001 (b) 11010 (c) 10110 (d) 11011 (e) 11100 −11010 −11001 −01101 −00111 −10101 1.38 Work Problem 1.37 for the following pairs of numbers: (a) 11010 (b) 01011 (c) 10001 (d) 10101 −10100 −11000 −01010 −11010 1.39 (a) A = 101010 and B = 011101 are 1’s complement numbers. Perform the following operations and indicate whether overflow occurs. (i) A + B (ii) A − B (b) Repeat part (a) assuming the numbers are 2’s complement numbers. 1.40 (a) Assume the integers below are 1’s complement integers. Find the 1’s complement of each number, and give the decimal values of the original number and of its complement. (i) 0000000 (ii) 1111111 (iii) 00110011 (iv) 1000000 (b) Repeat part (a) assuming the numbers are 2’s complement numbers and finding the 2’s complement of them. 1.41 An alternative algorithm for converting a base 20 integer, dn−1dn−2 · · · d1d0, into a base 10 integer is stated as follows: Multiply di by 2i and add i 0’s on the right, and then add all of the results. 28 Unit 1 (a) Use this algorithm to convert GA720 to base 10. (G20 is 1610.) (b) Prove that this algorithm is valid. (c) Consider converting a base 20 fraction, 0.d −1d −2 · · · d −n+1d −n, into a base 10 fraction. State an algorithm analogous to the one above for doing the conversion. (d) Apply your algorithm of part (c) to 0.FA720. 1.42 Let A and B be positive integers, and consider the addition of A and B in an n-bit 2’s complement number system. (a) Show that the addition of A and B produces the correct representation of the sum if the magnitude of (A + B) is < 2n−1 − 1 but it produces a representation of a negative number of magnitude 2n − (A + B) if the magnitude of (A + B) is > 2n−1 − 1. (b) Show that the addition of A and (−B) always produces the correct representation of the sum. Consider both the case where A ≥ B and the case A < B. (c) Show that the addition of (2n − A) + (2n − B), with the carry from the sign position ignored, produces the correct 2’s complement representation of −(A + B) if the magnitude of A + B is less than or equal to 2n−1. Also, show that it produces an incorrect sum representing the positive number 2n − (A + B) if the magnitude of (A + B) > 2n−1. 1.43 Let A and B be integers and consider the addition of A and B in an n-bit 1’s complement number system. Prove that addition of A and B using the end-around carry produces the correct representation of the sum provided overflow does not occur. Consider the four cases: A and B both positive, A positive and B negative with the magnitude of A greater than the magnitude of B, A positive and B negative with the magnitude of A less than or equal to the magnitude of B, and A and B both negative. 1.44 Prove that in a 2’s complement number system addition overflows if and only if the carry from the sign position does not equal the carry into the sign position. Consider the three cases: adding two positive numbers, adding two negative numbers, and adding two numbers of opposite sign. 1.45 Restate the method for detecting overflow of Problem 1.44 so that it applies to 1’s complement numbers. 1.46 Let B = bn−1bn−2 · · · b1b0 be an n-bit 2’s complement integer. Show that the decimal value of B is −bn−12n−1 + bn−22n−2 + bn−32n−3 + · · · + b12 + b0. (Hint: Consider positive (bn−1 = 0) and negative (bn−1 = 1) numbers separately, and note that the magnitude of a negative number is obtained by subtracting each bit from 1 (i.e., complementing each bit) and adding 1 to the result.) www.allitebooks.com UNIT Boolean Algebra 2 Objectives A list of some of the laws of switching algebra, which is a special case of Boolean algebra, is given in Table 2-3. Additional theorems of Boolean algebra are given in Table 2-4. When you complete this unit, you should be familiar with and be able to use these laws and theorems of Boolean algebra. 1. Understand the basic operations and laws of Boolean algebra. 2. Relate these operations and laws to circuits composed of AND gates, OR gates, and INVERTERS. Also relate these operations and laws to circuits composed of switches. 3. Prove any of these laws in switching algebra using a truth table. 4. Apply these laws to the manipulation of algebraic expressions including: a. Multiplying out an expression to obtain a sum of products (SOP) b. Factoring an expression to obtain a product of sums (POS) c. Simplifying an expression by applying one of the laws d. Finding the complement of an expression 29 Unit 2 Study Guide 1. In this unit you will study Boolean algebra, the basic mathematics needed for the logic design of digital systems. Just as when you first learned ordinary algebra, you will need a fair amount of practice before you can use Boolean algebra effectively. However, by the end of the course, you should be just as comfortable with Boolean algebra as with ordinary algebra. Fortunately, many of the rules of Boolean algebra are the same as for ordinary algebra, but watch out for some surprises! 2. Study Sections 2.1 and 2.2, Introduction and Basic Operations. (a) How does the meaning of the symbols 0 and 1 as used in this unit differ from the meaning as used in Unit 1? for AND ... (b) Two commonly used notations for the inverse or complement of A are A and A′. The latter has the advantage that it is much easier for typists, printers, and computers. (Have you ever tried to get a computer to print a bar over a letter?) We will use A′ for the complement of A. You may use either notation in your work, but please do not mix notations in the same equation. Most engineers use +GPS03BOEr PSOPTZNCPM GPS"/% BOEXF will follow this practice. An alternative notation, often used by mathematicians, is ∨ for OR and ∧ for AND. (c) Many different symbols are used for AND, OR, and INVERTER logic blocks. Initially we will use ... 30 for OR + for INVERTER The shapes of these symbols conform to those commonly used in industrial practice. We have added the +BOErGPSDMBSJUZ5IFTFTZNCPMTQPJOUJOUIF direction of signal flow. This makes it easier to read the circuit diagrams in comparison with the square or round symbols used in some books. (d) Determine the output of each of the following gates: 1 1 + 1 1 0 0 1 + 1 (e) Determine the unspecified inputs to each of the following gates if the outputs are as shown: 1 + 0 1 0 0 + 1 Boolean Algebra 3. 31 Study Section 2.3, Boolean Expressions and Truth Tables. (a) How many variables does the following expression contain? How many literals? A′BC′D + AB + B′CD + D′ (b) For the following circuit, if A = B = 0 and C = D = E = 1, indicate the output of each gate (0 or 1) on the circuit diagram: C D + A + F B E (c) Derive a Boolean expression for the circuit output. Then substitute A = B = 0 and C = D = E = 1 into your expression and verify that the value of F obtained in this way is the same as that obtained on the circuit diagram in (b). (d) Write an expression for the output of the following circuit and complete the truth table: AB A A′ A′ B (A′B)′ F B F= (e) When filling in the combinations of values for the variables on the left side of a truth table, always list the combinations of 0’s and 1’s in binary order. For example, for a three-variable truth table, the first row should be 000, the next row 001, then 010, 011, 100, 101, 110, and 111. Write an expression for the output of the following circuit and complete the truth table: A B ABC + C B′ A + B′ C(A + B′) F F= (f) Draw a gate circuit which has an output Z = [BC′ + F(E + AD′)] ′ (Hint: Start with the innermost parentheses and draw the circuit for AD′ first.) 32 Unit 2 Study Section 2.4, Basic Theorems. 4. (a) Prove each of the Theorems (2-4) through (2-8D) by showing that it is valid for both X = 0 and X = 1. (b) Determine the output of each of these gates: A A′ A A A A 0 1 A A + A′ A A + A + 0 1 + (c) State which of the basic theorems was used in simplifying each of the following expressions: (AB′ + C) · 0 = 0 A(B + C′) + 1 = 1 (BC′ + A)(BC′ + A) = BC′ + A X(Y′ + Z) + [X(Y′ + Z)] ′ = 1 (X′ + YZ)(X′ + YZ)′ = 0 D′(E′ + F) + D′(E′ + F) = D′(E′ + F) Study Section 2.5, Commutative, Associative, Distributive, and DeMorgan’s Laws. 5. (a) State the associative law for OR. (b) State the commutative law for AND. (c) Simplify the following circuit by using the associative laws. Your answer should require only two gates. A B + C G D E F + (d) For each gate determine the value of the unspecified input(s): + 0 1 1 1 0 0 0 + 1 0 (e) Using a truth table, verify the distributive law, Equation (2-11). 1 Boolean Algebra 33 (f) Illustrate the distributive laws, Equations (2-11) and (2-11D), using AND and OR gates. (g) Verify Equation (2-3) using the second distributive law. (h) Show how the second distributive law can be used to factor RS + T′ . 6. Study Section 2.6, Simplification Theorems. (a) By completing the truth table, prove that XY′ + Y = X + Y. XY 0 0 0 1 1 0 1 1 XY′ XY′ + Y X+Y (b) Which one of Theorems in Table 2-4 was applied to simplify each of the following expressions? Identify X and Y in each case. (A + B)(DE)′ + DE = A + B + DE AB′ + AB′C′D = AB′ (A′ + B)(CD + E′) + (A′ + B)(CD + E′)′ = A′ + B (A + BC′ + D′E)(A + D′E) = A + D′E 34 Unit 2 (c) Simplify the following circuit to a single gate: A B + C Z C + D (d) Work Problems 2.1, 2.2, 2.3, and 2.4. 7. Study Section 2.7, Multiplying Out and Factoring. (a) Indicate which of the following expressions are in the product-of-sums form, sum-of-products form, or neither: AB′ + D′EF ′ + G (A + B′C′)(A′ + BC) AB′(C′ + D + E′)(F ′ + G) X′Y + WX(X′ + Z) + A′B′C′ Your answer should indicate one expression as a product-of-sums form, one as sum-of-products form, and two as neither, not necessarily in that order. (b) When multiplying out an expression, why should the second distributive law be applied before the ordinary distributive law when possible? (c) Factor as much as possible using the ordinary distributive law: AD + B′CD + B′DE Now factor your result using the second distributive law to obtain a product of sums. (d) Work Problems 2.5, 2.6, and 2.7. 8. Probably the most difficult part of the unit is using the second distributive law for factoring or multiplying out an expression. If you have difficulty with Problems 2.5 or 2.6, or you cannot work them quickly, study the examples in Section 2.7 again, and then work the following problems. Multiply out: (a) (B′ + D + E)(B′ + D + A)(AE + C′) (b) (A + C′)(B′ + D)(C′ + D′)(C + D)E Boolean Algebra 35 As usual, when we say multiply out, we do not mean to multiply out by brute force, but rather to use the second distributive law whenever you can to cut down on the amount of work required. The answer to (a) should be of the following form: XX + XX + XX and (b) of the form: XXX + XXXXX, where each X represents a single variable or its complement. Now factor your answer to (a) to see that you can get back the original expression. 9. 10. Study Section 2.8, Complementing Boolean Expressions. Find the complement of each of the following expressions as indicated. In your answer, the complement operation should be applied only to single variables. (a) (ab′c′)′ = (b) (a′ + b + c + d′)′ = (c) (a′ + bc)′ = (d) (a′b′ + cd)′ = (e) [a(b′ + c′d)] ′ = 11. Because (X′)′ = X, if you complement each of your answers to 10, you should get back the original expression. Verify that this is true. (a) (b) (c) (d) (e) 12. Given that F = a′b + b′c, F′ = Complete the following truth table and verify that your answer is correct: abc 000 001 010 011 100 101 110 111 a′b b′c a′b + b′c (a + b′) (b + c′) F′ 36 Unit 2 13. A fully simplified expression should have nothing complemented except the individual variables. For example, F = (X + Y)′ (W + Z) is not a minimum product of sums. Find the minimum product of sums for F. 14. Work Problems 2.8 and 2.9. 15. Find the dual of (M + N′) P′. 16. Review the laws of Table 2-3 and the first three theorems of Table 2-4. Make sure that you can recognize when to apply them even if an expression has been substituted for a variable. 17. Reread the objectives of this unit. If you are satisfied that you can meet these objectives, take the readiness test. [Note: You will be provided with a copy of Tables 2-3 and 2-4 when you take the readiness test this time. However, by the end of Unit 3, you should know all the laws and theorems by memory.] Boolean Algebra 2.1 Introduction The basic mathematics needed for the study of logic design of digital systems is Boolean algebra. George Boole developed Boolean algebra in 1847 and used it to solve problems in mathematical logic. Boolean algebra has many other applications, including set theory and mathematical logic; however, we primarily consider its application to switching circuits. All of the switching devices we will use are essentially twostate devices (e.g., switches which are open or closed and transistors with high or low Boolean Algebra 37 output voltages). Consequently, we will emphasize the special case of Boolean algebra in which all of the variables assume only one of two values; this two-valued Boolean algebra is often called switching algebra. Claude Shannon first applied Boolean algebra to the design of switching circuits in 1939. First, we develop some of the properties of switching algebra and use these to define a general Boolean algebra. We will use a Boolean variable, such as X or Y, to represent the input or output of a switching circuit. We will assume that each of these variables can take on only two different values. The symbols “0” and “1” are used to represent these two different values. Thus, if X is a Boolean (switching) variable, then either X = 0 or X = 1. The symbols “0” and “1” used in Boolean algebra do not have a numeric value; instead they represent two different states in a logic circuit and are the two values of a switching variable. In a logic gate circuit, 0 (usually) represents a range of low voltages, and 1 represents a range of high voltages. In a switch circuit, 0 (usually) represents an open switch, and 1 represents a closed circuit. In general, 0 and 1 can be used to represent the two states in any binary-valued system. 2.2 Basic Operations The basic operations of Boolean (switching) algebra are called AND, OR, and complement (or inverse). In the case of switch circuits these operations correspond to different configurations of switches. To apply switching algebra to a switch circuit, each switch contact is labeled with a variable. If contact X is open, the variable X is defined to be 0; if contact X is closed, the variable X is defined to be 1. X X = 0 → switch open X = 1 → switch closed The contacts in a switch can be normally open (NO) or normally closed (NC). When the switch position is changed, the NO contact closes and the NC contact opens, so the NO and NC contacts are always in opposite states. If X is the variable assigned to the NO contact, then the variable assigned to the NC contact is the complement of X, denoted as X′ , where the prime (′) denotes complementation. A B NO contact NC contact The complement of 0 is 1, and the complement of 1 is 0. Symbolically, we write 0′ = 1 and 1′ = 0 38 Unit 2 If Xis a switching variable, X′ = 1 if X = 0 and X′ = 0 if X = 1 An alternate name for complementation is inversion, and the electronic circuit which forms the inverse of X is referred to as an inverter. Symbolically, we represent an inverter by X X′ where the circle at the output indicates inversion. A low voltage at the inverter input produces a high voltage at the output and vice versa. In a general switch circuit, the value 0 is assigned to the connection between two terminals in the circuit if there is no connection (open circuit) between the terminals, and a 1 is assigned if there is a connection (closed circuit) between the terminals. If the switch circuit only contains two switches, the switch contacts must be connected in series or in parallel. When switch contacts A and B are connected in series, there is an open circuit between the terminals if either A or B or both are open (0), and there is a closed circuit between the terminals only if both A and B are closed (1). 1 A B 2 C = 0 → open circuit between terminals 1 and 2 C = 1 → closed circuit between terminals 1 and 2 This is summarized in the following truth table: AB 0 0 0 1 1 0 1 1 C=A·B 0 0 0 1 The operation defined by the table is called AND and it is written algebraically as C = A · B. The “·” symbol is frequently omitted in a Boolean expression, and we will usually write AB instead of A · B. The AND operation is also referred to as logical (or Boolean) multiplication. When switches A and B are connected in parallel, there is a closed circuit between the terminals if either A or B is closed (1), and there is an open circuit between the terminals only if both A and B are open (0). A 1 B www.allitebooks.com 2 Boolean Algebra 39 This is summarized in the following truth table: AB 0 0 0 1 1 0 1 1 C=A+B 0 1 1 1 The operation defined by the table is called OR and it is written algebraically as C = A + B. This type of OR operation is sometimes referred to as inclusive OR as opposed to exclusive OR, which is defined later. The OR operation is also referred to as logical (or Boolean) addition. Logic gates operate so that the voltage on inputs and outputs of a gate is either in a low voltage range or a high voltage range, except when the signals are changing. Switching algebra can be applied to logic gates by assigning 0 and 1 to the two voltage ranges. Usually, a 0 is assigned to the low voltage range and a 1 to the high voltage range. A logic gate which performs the AND operation is represented by A C=AB B The gate output is C = 1 if and only if the gate inputs A = 1 and B = 1. A logic gate which performs the OR operation is represented by A B + C=A+B The gate output is C = 1 if and only if the gate inputs A = 1 or B = 1 (or both). Electronic circuits which realize inverters and AND and OR gates are described in Appendix A. 2.3 Boolean Expressions and Truth Tables Boolean expressions are formed by application of the basic operations to one or more variables or constants. The simplest expressions consist of a single constant or variable, such as 0, X, or Y′. More complicated expressions are formed by combining two or more other expressions using AND or OR, or by complementing another expression. Examples of expressions are AB′ + C [A(C + D)] ′ + BE (2-1) (2-2) Parentheses are added as needed to specify the order in which the operations are performed. When parentheses are omitted, complementation is performed first followed by AND and then OR. Thus in Expression (2-1), B′ is formed first, then AB′, and finally AB′ + C. 40 Unit 2 Each expression corresponds directly to a circuit of logic gates. Figure 2-1 gives the circuits for Expressions (2-1) and (2-2). A FIGURE 2-1 Circuits for Expressions (2-1) and (2-2) B AB′ C B′ + (AB′ + C) (a) © Cengage Learning 2014 C D (C + D) + A(C + D) [A(C + D)]′ + A B E [A(C + D)]′ + BE BE (b) An expression is evaluated by substituting a value of 0 or 1 for each variable. If A = B = C = 1 and D = E = 0, the value of Expression (2-2) is [A(C + D)] ′ + BE = [1(1 + 0)] ′ + 1 · 0 = [1(1)] ′ + 0 = 0 + 0 = 0 Each appearance of a variable or its complement in an expression will be referred to as a literal. Thus, the following expression, which has three variables, has 10 literals: ab′c + a′b + a′bc′ + b′c′ When an expression is realized using logic gates, each literal in the expression corresponds to a gate input. A truth table (also called a table of combinations) specifies the values of a Boolean expression for every possible combination of values of the variables in the expression. The name truth table comes from a similar table which is used in symbolic logic to list the truth or falsity of a statement under all possible conditions. We can use a truth table to specify the output values for a circuit of logic gates in terms of the values of the input variables. The output of the circuit in Figure 2-2(a) is F = A′ + B. Figure 2-2(b) shows a truth table which specifies the output of the circuit for all possible combinations of values of the inputs A and B. The first two columns list the four combinations of values of A and B, and the next column gives the corresponding values of A′. The last column, which gives the values of A′ + B, is formed by ORing together corresponding values of A′ and B in each row. FIGURE 2-2 Two-Input Circuit and Truth Table © Cengage Learning 2014 A A′ B + (a) F = A′ + B A 0 0 1 (b) 1 B 0 1 0 1 A′ 1 1 0 0 F = A′ + B 1 1 0 1 Boolean Algebra 41 Next, we will use a truth table to specify the value of Expression (2-1) for all possible combinations of values of the variables A, B, and C. On the left side of Table 2-1, we list the values of the variables A, B, and C. Because each of the three variables can assume the value 0 or 1, there are 2 × 2 × 2 = 8 combinations of values of the variables. These combinations are easily obtained by listing the binary numbers 000, 001, ... , 111. In the next three columns of the truth table, we compute B′ , AB′, and AB′ + C, respectively. Two expressions are equal if they have the same value for every possible combination of the variables. The expression (A + C)(B′ + C) is evaluated using the last three columns of Table 2-1. Because it has the same value as AB′ + C for all eight combinations of values of the variables A, B, and C, we conclude TABLE 2-1 © Cengage Learning 2014 A 0 0 0 0 1 1 1 1 B 0 0 1 1 0 0 1 1 C 0 1 0 1 0 1 0 1 B′ 1 1 0 0 1 1 0 0 AB′ 0 0 0 0 1 1 0 0 AB′ + C 0 1 0 1 1 1 0 1 A+C 0 1 0 1 1 1 1 1 B′ + C 1 1 0 1 1 1 0 1 (A + C)(B′ + C) 0 1 0 1 1 1 0 1 AB′ + C = (A + C)(B′ + C) (2-3) If an expression has n variables, and each variable can have the value 0 or 1, the number of different combinations of values of the variables is 2 × 2 × 2 × . . . = 2n n times Therefore, a truth table for an n-variable expression will have 2n rows. 2.4 Basic Theorems The following basic laws and theorems of Boolean algebra involve only a single variable: Operations with 0 and 1: X+0=X (2-4) X·1=X (2-4D) X+1=1 (2-5) X·0=0 (2-5D) 42 Unit 2 Idempotent laws: X+X=X (2-6) Involution law: (X′)′ = X (2-7) Laws of complementarity: X + X′ = 1 (2-8) X·X=X (2-6D) X · X′ = 0 (2-8D) Each of these theorems is easily proved by showing that it is valid for both of the possible values of X. For example, to prove X + X′ = 1, we observe that if X = 0, 0 + 0′ = 0 + 1 = 1, and if X = 1, 1 + 1′ = 1 + 0 = 1 Any expression can be substituted for the variable X in these theorems. Thus, by Theorem (2-5), (AB′ + D)E + 1 = 1 and by Theorem (2-8D), (AB′ + D)(AB′ + D)′ = 0 We will illustrate some of the basic theorems with circuits of switches. As before, 0 will represent an open circuit or open switch, and 1 will represent a closed circuit or closed switch. If two switches are both labeled with the variable A, this means that both switches are open when A = 0 and both are closed when A = 1. Thus the circuit A A can be replaced with a single switch: A This illustrates the theorem A · A = A. Similarly, A A = A which illustrates the theorem A + A = A. A switch in parallel with an open circuit is equivalent to the switch alone A = (A + 0 = A) A Boolean Algebra 43 while a switch in parallel with a short circuit is equivalent to a short circuit. A = (A + 1 = 1) If a switch is labeled A′, then it is open when A is closed and conversely. Hence, A in parallel with A′ can be replaced with a closed circuit because one or the other of the two switches is always closed. A = A′ (A + A′ = 1)

such as adding xx′, multiplying by (x + x′), adding yz to xy + x′z, or adding xy to x. When possible, the added terms should be chosen so that they will combine with or eliminate other terms. Example WX + XY + X′Z′ + WY′Z′ = WX + XY + X′Z′ + WY′Z′ + WZ′ = WX + XY + X′Z′ + WZ′ = WX + XY + X′Z′ (add WZ′ by consensus theorem) (eliminate WY′Z′) (eliminate WZ′) (3-27) The following comprehensive example illustrates the use of all four methods: Example A′B′C′D′ + A′BC′D′ + A′BD + A′BC′D + ABCD + ACD′ + B′CD′ ① A′C′D′ ② = A′C′D′ + BD(A′ + AC) + ACD′ + B′CD′ ③ = A′C′D′ + A′BD + BCD + ACD′ + B′CD′ + ABC ④ 74 Unit 3 consensus ACD′ = A′C′D′ + A′BD + BCD + ACD′ + B′CD′ + ABC consensus BCD = A′C′D′ + A′BD + B′CD′ + ABC (3-28) What theorems were used in steps 1, 2, 3, and 4? If the simplified expression is to be left in a product-of-sums form instead of a sum-of-products form, the duals of the preceding theorems should be applied. (A′ + B′ + C′)(A′ + B′ + C)(B′ + C)(A + C)(A + B + C) ② ① (A′ + B′) = (A′ + B′)(B′ + C)(A + C) = (A′ + B′)(A + C) Example ③ (3-29) What theorems were used in steps 1, 2, and 3? In general, there is no easy way of determining when a Boolean expression has a minimum number of terms or a minimum number of literals. Systematic methods for finding minimum sum-of-products and minimum product-of-sums expressions will be discussed in Units 5 and 6. 3.5 Proving Validity of an Equation Often we will need to determine if an equation is valid for all combinations of values of the variables. Several methods can be used to determine if an equation is valid: 1. 2. 3. 4. Construct a truth table and evaluate both sides of the equation for all combinations of values of the variables. (This method is rather tedious if the number of variables is large, and it certainly is not very elegant.) Manipulate one side of the equation by applying various theorems until it is identical with the other side. Reduce both sides of the equation independently to the same expression. It is permissible to perform the same operation on both sides of the equation provided that the operation is reversible. For example, it is all right to complement both sides of the equation, but it is not permissible to multiply both sides of the equation by the same expression. (Multiplication is not reversible because division is not defined for Boolean algebra.) Similarly, it is not permissible to add the same term to both sides of the equation because subtraction is not defined for Boolean algebra. Boolean Algebra (Continued) 75 To prove that an equation is not valid, it is sufficient to show one combination of values of the variables for which the two sides of the equation have different values. When using method 2 or 3 above to prove that an equation is valid, a useful strategy is to 1. 2. 3. 4. First reduce both sides to a sum of products (or a product of sums). Compare the two sides of the equation to see how they differ. Then try to add terms to one side of the equation that are present on the other side. Finally try to eliminate terms from one side that are not present on the other. Whatever method is used, frequently compare both sides of the equation and let the difference between them serve as a guide for what steps to take next. Example 1 Show that A′BD′ + BCD + ABC′ + AB′D = BC′D′ + AD + A′BC Starting with the left side, we first add consensus terms, then combine terms, and finally eliminate terms by the consensus theorem. A′BD′ + BCD + ABC′ + AB′D = A′BD′ + BCD + ABC′ + AB′D + BC′D′ + A′BC + ABD (add consensus of A′BD′ and ABC′) (add consensus of A′BD′ and BCD) (add consensus of BCD and ABC′) = AD + A′BD′ + BCD + ABC′ + BC′D′ + A′BC = BC′D′ + AD + A′BC (eliminate consensus of BC′D′ and AD) (eliminate consensus of AD and A′BC) (eliminate consensus of BC′D′ and A′BC) (3-30) Example 2 Show that the following equation is valid: A′BC′D + (A′ + BC)(A + C′D′) + BC′D + A′BC′ = ABCD + A′C′D′ + ABD + ABCD′ + BC′D First, we will reduce the left side: A′BC′D + (A′ + BC)(A + C′D′) + BC′D + A′BC′ (eliminate A′BC′D using absorption) = (A′ + BC)(A + C′D′) + BC′D + A′BC′ (multiply out using (3-3)) = ABC + A′C′D′ + BC′D + A′BC′ (eliminate A′BC′ by consensus) = ABC + A′C′D′ + BC′D 76 Unit 3 Now we will reduce the right side: = ABCD + A′C′D′ + ABD + ABCD′ + BC′D (combine ABCD and ABCD′) = ABC + A′C′D′ + ABD + BC′D (eliminate ABD by consensus) = ABC + A′C′D′ + BC′D Because both sides of the original equation were independently reduced to the same expression, the original equation is valid. As we have previously observed, some of the theorems of Boolean algebra are not true for ordinary algebra. Similarly, some of the theorems of ordinary algebra are not true for Boolean algebra. Consider, for example, the cancellation law for ordinary algebra: If x + y = x + z, then y=z (3-31) The cancellation law is not true for Boolean algebra. We will demonstrate this by constructing a counterexample in which x + y = x + z but y = z. Let x = 1, y = 0, z = 1. Then, 1 + 0 = 1 + 1 but 0 ≠ 1 In ordinary algebra, the cancellation law for multiplication is If xy = xz, then y=z (3-32) This law is valid provided x ≠ 0. In Boolean algebra, the cancellation law for multiplication is also not valid when x = 0. (Let x = 0, y = 0, z = 1; then 0 · 0 = 0 · 1, but 0 ≠ 1). Because x = 0 about half of the time in switching algebra, the cancellation law for multiplication cannot be used. Even though Statements (3-31) and (3-32) are generally false for Boolean algebra, the converses If y = z, If y = z, then then x+y=x+z xy = xz (3-33) (3-34) are true. Thus, we see that although adding the same term to both sides of a Boolean equation leads to a valid equation, the reverse operation of canceling or subtracting a term from both sides generally does not lead to a valid equation. Similarly, multiplying both sides of a Boolean equation by the same term leads to a valid equation, but not conversely. When we are attempting to prove that an equation is valid, it is not permissible to add the same expression to both sides of the equation or to multiply both sides by the same expression, because these operations are not reversible. Boolean Algebra (Continued) 77 Programmed Exercise 3.1 Cover the answers to this exercise with a sheet of paper and slide it down as you check your answers. Write your answer in the space provided before looking at the correct answer. The following expression is to be multiplied out to form a sum of products: (A + B + C′)(A′ + B′ + D)(A′ + C + D′)(A + C′ + D) First, find a pair of sum terms which have two literals in common and apply the second distributive law. Also, apply the same law to the other pair of terms. Answer (A + C′ + BD) [A′ + (B′ + D)(C + D′)] (Note: This answer was obtained by using (X + Y)(X + Z) = X + YZ.) Next, find a pair of sum terms which have a variable in one and its complement in the other. Use the appropriate theorem to multiply these sum terms together without introducing any redundant terms. Apply the same theorem a second time. Answer (A + C′ + BD)(A′ + B′D′ + CD) = A(B′D′ + CD) + A′(C′ + BD) or A(B′ + D)(C + D′) + A′(C′ + BD) (Note: This answer was obtained using (X + Y)(X′ + Z) = XZ + X′Y.) Complete the problem by multiplying out using the ordinary distributive law. Final Answer AB′D′ + ACD + A′C′ + A′BD Programmed Exercise 3.2 Cover the answers to this exercise with a sheet of paper and slide it down as you check your answers. Write your answer in the space provided before looking at the correct answer. The following expression is to be factored to form a product of sums: WXY′ + W′X′Z + WY′Z + W′YZ′ First, factor as far as you can using the ordinary distributive law. 78 Unit 3 Answer WY′(X + Z) + W′(X′Z + YZ′) Next, factor further by using a theorem which involves a variable and its complement. Apply this theorem twice. (W + X′Z + YZ′) [W′ + Y′(X + Z)] = [W + (X′ + Z′)(Y + Z)][W′ + Y′(X + Z)] Answer or WY′(X + Z) + W′(X′ + Z′)(Y + Z) = [W + (X′ + Z′)(Y + Z)][W′ + Y′(X + Z)] [Note: This answer was obtained by using AB + A′C = (A + C)(A′ + B).] Now, complete the factoring by using the second distributive law. Final answer (W + X′ + Z′)(W + Y + Z)(W′ + Y′)(W′ + X + Z) Programmed Exercise 3.3 Cover the answers to this exercise with a sheet of paper and slide it down as you check your answers. Write your answer in the space provided before looking at the correct answer. The following expression is to be simplified using the consensus theorem: AC′ + AB′D + A′B′C + A′CD′ + B′C′D′ First, find all of the consensus terms by checking all pairs of terms. Answer The consensus terms are indicated. A′B′D′ AC′ + AB′D + A′B′C + A′CD′ + B′C′D′ B′CD AB′C′ A′B′D′ Boolean Algebra (Continued) 79 Can the original expression be simplified by the direct application of the consensus theorem? Answer No, because none of the consensus terms appears in the original expression. Now add the consensus term B′CD to the original expression. Compare the added term with each of the original terms to see if any consensus exists. Eliminate as many of the original terms as you can. Answer (AB′D) AC′ + AB′D + A′B′C + A′CD′ + B′C′D′ + B′CD (A′B′C) Now that we have eliminated two terms, can B′CD also be eliminated? What is the final reduced expression? Answer No, because the terms used to form B′CD are gone. Final answer is AC′ + A′CD′ + B′C′D′ + B′CD Programmed Exercise 3.4 Keep the answers to this exercise covered with a sheet of paper and slide it down as you check your answers. Problem: The following expression is to be simplified ab′cd′e + acd + acf′gh′ + abcd′e + acde′ + e′h′ State a theorem which can be used to combine a pair of terms and apply it to combine two of the terms in the above expression. Answer Apply XY + XY′ = X to the terms ab′cd′e and abcd′e, which reduces the expression to acd′e + acd + acf′gh′ + acde′ + e′h′ 80 Unit 3 Now state a theorem (other than the consensus theorem) which can be used to eliminate terms and apply it to eliminate a term in this expression. Answer Apply X + XY = X to eliminate acde′. (What term corresponds toX?) The result is acd′e + acd + acf′gh′ + e′h′ Now state a theorem that can be used to eliminate literals and apply it to eliminate a literal from one of the terms in this expression. (Hint: It may be necessary to factor out some common variables from a pair of terms before the theorem can be applied.) Answer Use X + X′Y = X + Y to eliminate a literal from acd′e. To do this, first factor ac out of the first two terms: acd′e + acd = ac(d + d′e). After eliminating d′, the resulting expression is ace + acd + acf′gh′ + e′h′ (a) Can any term be eliminated from this expression by the direct application of the consensus theorem? (b) If not, add a redundant term using the consensus theorem, and use this redundant term to eliminate one of the other terms. (c) Finally, reduce your expression to three terms. Answer (a) No (b) Add the consensus of ace and e′h′: ace + acd + acf′gh′ + e′h′ + ach′ Now eliminate acf′gh′ (by X + XY = X) ace + acd + e′h′ + ach′ (c) Now eliminate ach′ by the consensus theorem. The final answer is ace + acd + e′h′ Boolean Algebra (Continued) 81 Programmed Exercise 3.5 Keep the answers to this exercise covered with a sheet of paper and slide it down as you check your answers. Z = (A + C′ + F ′ + G)(A + C′ + F + G)(A + B + C′ + D′ + G) (A + C + E + G)(A′ + B + G)(B + C′ + F + G) This is to be simplified to the form (X + X + X)(X + X + X)(X + X + X) where each X represents a literal. State a theorem which can be used to combine the first two sum terms of Z and apply it. (Hint: The two sum terms differ in only one variable.) Answer (X + Y)(X + Y′) = X Z = (A + C′ + G)(A + B + C′ + D′ + G)(A + C + E + G)(A′ + B + G) (B + C′ + F + G) Now state a theorem (other than the consensus theorem) which can be used to eliminate a sum term and apply it to this expression. Answer X(X + Y) = X Z = (A + C′ + G)(A + C + E + G)(A′ + B + G)(B + C′ + F + G) Next, eliminate one literal from the second term, leaving the expression otherwise unchanged. (Hint: This cannot be done by the direct application of one theorem; it will be necessary to partially multiply out the first two sum terms before eliminating the literal.) Answer (A + C′ + G)(A + C + E + G) = A + G + C′(C + E) = A + G + C′E Therefore, Z = (A + C′ + G)(A + E + G)(A′ + B + G)(B + C′ + F + G) 82 Unit 3 (a) Can any term be eliminated from this expression by the direct application of the consensus theorem? (b) If not, add a redundant sum term using the consensus theorem, and use this redundant term to eliminate one of the other terms. (c) Finally, reduce your expression to a product of three sum terms. Answer (a) No (b) Add B + C′ + G (consensus of A + C′ + G and A′ + B + G). Use X(X + Y) = X, where X = B + C′ + G, to eliminate B + C′ + F + G. (c) Now eliminate B + C′ + G by consensus. The final answer is Z = (A + C′ + G)(A + E + G)(A′ + B + G) Problems 3.6 In each case, multiply out to obtain a sum of products: (Simplify where possible.) (a) (W + X′ + Z′)(W′ + Y′)(W′ + X + Z′)(W + X′)(W + Y + Z) (b) (A + B + C + D)(A′ + B′ + C + D′)(A′ + C)(A + D)(B + C + D) 3.7 Factor to obtain a product of sums. (Simplify where possible.) (a) BCD + C′D′ + B′C′D + CD (b) A′C′D′ + ABD′ + A′CD + B′D 3.8 Write an expression for F and simplify. A B ⊕ F A D 3.9 D + Is the following distributive law valid? A ⊕ BC = (A ⊕ B)(A ⊕ C) Prove your answer. 3.10 (a) Reduce to a minimum sum of products (three terms): (X + W)(Y ⊕ Z) + XW′ (b) Reduce to a minimum sum of products (four terms): (A ⊕ BC) + BD + ACD (c) Reduce to a minimum product of sums (three terms): (A′ + C′ + D′)(A′ + B + C′)(A + B + D)(A + C + D) Boolean Algebra (Continued) 83 3.11 Simplify algebraically to a minimum sum of products (five terms): (A + B′ + C + E′)(A + B′ + D′ + E)(B′ + C′ + D′ + E′) 3.12 Prove algebraically that the following equation is valid: A′CD′E + A′B′D′ + ABCE + ABD = A′B′D′ + ABD + BCD′E 3.13 Simplify each of the following expressions: (a) KLMN′ + K′L′MN + MN′ (b) KL′M′ + MN′ + LM′N′ (c) (K + L′)(K′ + L′ + N)(L′ + M + N′) (d) (K′ + L + M′ + N)(K′ + M′ + N + R)(K′ + M′ + N + R′)KM 3.14 Factor to obtain a product of sums: (a) K′L′M + KM′N + KLM + LM′N′ (b) KL + K′L′ + L′M′N′ + LMN′ (c) KL + K′L′M + L′M′N + LM′N′ (d) K′M′N + KL′N′ + K′MN′ + LN (e) WXY + WX′Y + WYZ + XYZ′ (four terms) (four terms) (four terms) (four terms) (three terms) 3.15 Multiply out to obtain a sum of products: (a) (K′ + M′ + N)(K′ + M)(L + M′ + N′)(K′ + L + M)(M + N) (three terms) (b) (K′ + L′ + M′)(K + M + N′)(K + L)(K′ + N)(K′ + M + N) (c) (K′ + L′ + M)(K + N′)(K′ + L + N′)(K + L)(K + M + N′) (d) (K + L + M)(K′ + L′ + N′)(K′ + L′ + M′)(K + L + N) (e) (K + L + M)(K + M + N)(K + L′ + M′)(K′ + M′ + N′) 3.16 Eliminate the exclusive OR, and then factor to obtain a minimum product of sums: (a) (KL ⊕ M) + M′N′ (b) M′(K ⊕ N′) + MN + K′N 3.17 Algebraically prove identities involving the equivalence (exclusive-NOR) operation: (a) x ≡ 0 = x′ (b) x ≡ 1 = x (c) x ≡ x = 1 (d) x ≡ x′ = 0 (e) x ≡ y = y ≡ x (f) (x ≡ y) ≡ z = x ≡ (y ≡ z) (g) (x ≡ y)′ = x′ ≡ y = x ≡ y′ 3.18 Algebraically prove identities involving the exclusive-OR operation: (a) x ⊕ 0 = x (b) x ⊕ 1 = x′ (c) x ⊕ x = 0 (d) x ⊕ x′ = 1 (e) x ⊕ y = y ⊕ x (f) (x ⊕ y) ⊕ z = x ⊕ (y ⊕ z) (g) (x ⊕ y)′ = x′ ⊕ y = x ⊕ y′ 84 Unit 3 3.19 Algebraically prove the following identities: (a) x + y = x ⊕ y ⊕ xy (b) x + y = x ≡ y ≡ xy 3.20 Algebraically prove or disprove the following distributive identities: (a) x(y ⊕ z) = xy ⊕ xz (b) x + (y ⊕ z) = (x + y) ⊕ (x + z) (c) x(y ≡ z) = xy ≡ xz (d) x + (y ≡ z) = (x + y) ≡ (x + z) 3.21 Simplify each of the following expressions using only the consensus theorem (or its dual): (a) BC′D′ + ABC′ + AC′D + AB′D + A′BD′ (reduce to three terms) (b) W′Y′ + WYZ + XY′Z + WX′Y (reduce to three terms) (c) (B + C + D)(A + B + C)(A′ + C + D)(B′ + C′ + D′) (d) W′XY + WXZ + WY′Z + W′Z′ (e) A′BC′ + BC′D′ + A′CD + B′CD + A′BD (f) (A + B + C)(B + C′ + D)(A + B + D)(A′ + B′ + D′) 3.22 Factor Z = ABC + DE + ACF + AD′ + AB′E′ and simplify it to the form (X + X) (X + X)(X + X + X + X) (where each X represents a literal). Now express Z as a minimum sum of products in the form: XX + XX + XX + XX 3.23 Repeat Problem 3.22 for F = A′B + AC + BC′D′ + BEF + BDF. 3.24 Factor to obtain a product of four terms and then reduce to three terms by applying the consensus theorem: X′Y′Z′ + XYZ 3.25 Simplify each of the following expressions: (a) xy + x′yz′ + yz (b) (xy′ + z)(x + y′)z (c) xy′ + z + (x′ + y)z′ (d) a′d(b′ + c) + a′d′(b + c′) + (b′ + c)(b + c′) (e) w′x′ + x′y′ + yz + w′z′ (f) A′BCD + A′BC′D + B′EF + CDE′G + A′DEF + A′B′EF (reduce to a sum of three terms) (g) [(a′ + d′ + b′c)(b + d + ac′)] ′ + b′c′d′ + a′c′d (reduce to three terms) 3.26 Simplify to a sum of three terms: (a) A′C′D′ + AC′ + BCD + A′CD′ + A′BC + AB′C′ (b) A′B′C′ + ABD + A′C + A′CD′ + AC′D + AB′C′ 3.27 Reduce to a minimum sum of products: F = WXY′ + (W′Y′ ≡ X) + (Y ⊕ WZ) Boolean Algebra (Continued) 85 3.28 Determine which of the following equations are always valid (give an algebraic proof): (a) a′b + b′c + c′a = ab′ + bc′ + ca′ (b) (a + b)(b + c)(c + a) = (a′ + b′)(b′ + c′)(c′ + a′) (c) abc + ab′c′ + b′cd + bc′d + ad = abc + ab′c′ + b′cd + bc′d (d) xy′ + x′z + yz′ = x′y + xz′ + y′z (e) (x + y)(y + z)(x + z) = (x′ + y′)(y′ + z′)(x′ + z′) (f) abc′ + ab′c + b′c′d + bcd = ab′c + abc′ + ad + bcd + b′c′d 3.29 The following circuit is implemented using two half-adder circuits. The expressions for the half-adder outputs are S = A ⊕ B, and C = AB. Derive simplified sum-ofproducts expressions for the circuit outputs SUM and Co. Give the truth table for the outputs. X A S A S Y B C B C SUM Co Ci 3.30 The output of a majority circuit is 1 if a majority (more than half) of its inputs are equal to 1, and the output is 0 otherwise. Construct a truth table for a three-input majority circuit and derive a simplified sum-of-products expression for its output. 3.31 Prove algebraically: (a) (X′ + Y′)(X ≡ Z) + (X + Y)(X ⊕ Z) = (X ⊕ Y) + Z′ (b) (W′ + X + Y′)(W + X′ + Y)(W + Y′ + Z) = X′Y′ + WX + XYZ + W′YZ (c) ABC + A′C′D′ + A′BD′ + ACD = (A′ + C)(A + D′)(B + C′ + D) 3.32 Which of the following statements are always true? Justify your answers. (a) If A + B = C, then AD′ + BD′ = CD′ (b) If A′B + A′C = A′D, then B + C = D (c) If A + B = C, then A + B + D = C + D (d) If A + B + C = C + D, then A + B = D 3.33 Find all possible terms that could be added to each expression using the consensus theorem. Then reduce to a minimum sum of products. (a) A′C′ + BC + AB′ + A′BD + B′C′D′ + ACD′ (b) A′CD′ + BC′D + A′C′D + BC′D′ + AB′C′ + A′BC 3.34 Simplify the following expression to a sum of two terms and then factor the result to obtain a product of sums: abd′f′ + b′cegh′ + abd′f + acd′e + b′ce 3.35 Multiply out the following expression and simplify to obtain a sum-of-products expression with three terms: (a + c)(b′ + d)(a + c′ + d′)(b′ + c′ + d′) 86 Unit 3 3.36 Factor and simplify to obtain a product-of-sums expression with four terms: abc′ + d′e + ace + b′cd′ 3.37 (a) Show that x ⊕ y = (x ≡ y)′ (b) Realize a′b′c′ + a′bc + ab′c + abc′ using only two-input equivalence gates. 3.38 In a Boolean algebra, which of the following statements are true? Prove your answer. (a) If x(y + a′) = x(y + b′), then a = b. (b) If a′b + ab′ = a′c + ac′, then b = c. 3.39 The definition of Boolean algebra given in Unit 2 is redundant (i.e., not all of the properties are independent). For example, show that the associative property a + (b + c) = (a + b) + c can be proved using the other properties of Boolean algebra. (Hint: Consider expanding [a + (b + c)][(a + b) + c] in two different ways. Be sure to not use the associative property.) UNIT Applications of Boolean Algebra Minterm and Maxterm Expansions 4 Objectives 1. Given a word description of the desired behavior of a logic circuit, write the output of the circuit as a function of the input variables. Specify this function as an algebraic expression or by means of a truth table, as is appropriate. 2. Given a truth table, write the function (or its complement) as both a minterm expansion (standard sum of products) and a maxterm expansion (standard product of sums). Be able to use both alphabetic and decimal notation. 3. Given an algebraic expression for a function, expand it algebraically to obtain the minterm or maxterm form. 4. Given one of the following: minterm expansion for F, minterm expansion for F′, maxterm expansion for F, or maxterm expansion for F′, find any of the other three forms. 5. Write the general form of the minterm and maxterm expansion of a function of n variables. 6. Explain why some functions contain don't-care terms. 7. Explain the operation of a full adder and a full subtracter and derive logic equations for these modules. Draw a block diagram for a parallel adder or subtracter and trace signals on the block diagram. 87 88 Unit 4 Study Guide In the previous units, we placed a dot (·) inside the AND-gate symbol, a plus sign (+) inside the OR-gate symbol, and a ⊕ inside the exclusive OR. Because you are now familiar with the relationship between the shape of the gate symbol and the logic function performed, we will omit the · , + , and ⊕ and use the standard gate symbols for AND, OR, and exclusive OR in the rest of the book. 1. Study Section 4.1, Conversion of English Sentences to Boolean Equations. (a) Use braces to identify the phrases in each of the following sentences: (1) The tape reader should stop if the manual stop button is pressed, if an error occurs, or if an end-of-tape signal is present. (2) He eats eggs for breakfast if it is not Sunday and he has eggs in the refrigerator. (3) Addition should occur iff an add instruction is given and the signs are the same, or if a subtract instruction is given and the signs are not the same. (b) Write a Boolean expression which represents each of the sentences in (a). Assign a variable to each phrase, and use a complemented variable to represent a phrase which contains “not”. (Your answers should be in the form F = S′E, F = AB + SB′, and F = A + B + C, but not necessarily in that order.) (c) If X represents the phrase “N is greater than 3”, how can you represent the phrase “N is less than or equal to 3”? (d) Work Problems 4.1 and 4.2. 2. Study Section 4.2, Combinational Logic Design Using a Truth Table. Previously, you have learned how to go from an algebraic expression for a function to a truth table; in this section you will learn how to go from a truth table to an algebraic expression. (a) Write a product term which is 1 iff a = 0, b = 0, and c = 1. (b) Write a sum term which is 0 iff a = 0, b = 0, and c = 1. (c) Verify that your answers to (a) and (b) are complements. Applications of Boolean Algebra Minterm and Maxterm Expansions 89 (d) Write a product term which is 1 iff a = 1, b = 0, c = 0, and d = 1. (e) Write a sum term which is 0 iff a = 0, b = 0, c = 1, and d = 1. (f) For the given truth table, write F as a sum of four product terms which correspond to the four 1's of F. (g) From the truth table write F as a product of four sum terms which correspond to the four 0's of F. (h) Verify that your answers to both (f) and (g) reduce to F = b′c′ + a′c. 3. a 0 0 0 0 1 1 1 1 b 0 0 1 1 0 0 1 1 c 0 1 0 1 0 1 0 1 F 1 1 0 1 1 0 0 0 Study Section 4.3, Minterm and Maxterm Expansions. (a) Define the following terms: minterm (for n variables) maxterm (for n variables) (b) Study Table 4-1 and observe the relation between the values of A, B, and C and the corresponding minterms and maxterms. If A = 0, then does A or A′ appear in the minterm? In the maxterm? If A = 1, then does A or A′ appear in the minterm? In the maxterm? What is the relation between minterm, mi, and the corresponding maxterm, Mi? (c) For the table given in Study Guide Question 2(f), write the minterm expansion for F in m-notation and in decimal notation. For the same table, write the maxterm expansion for F in M-notation and in decimal notation. Check your answers by converting your answer to 2(f) to m-notation and your answer to 2(g) to M-notation. 90 Unit 4 (d) Given a sum-of-products expression, how do you expand it to a standard sum of products (minterm expansion)? (e) Given a product-of-sums expression, how do you expand it to a standard product of sums (maxterm expansion)? (f) In Equation (4-11), what theorems were used to factor f to obtain the maxterm expansion? (g) Why is the following expression not a maxterm expansion? f(A, B, C, D) = (A + B′ + C + D)(A′ + B + C′)(A′ + B + C + D′) (h) Assuming that there are three variables (A, B, C), identify each of the following as a minterm expansion, maxterm expansion, or neither: (1) AB + B′C′ (3) A + B + C (5) A′BC′ + AB′C + ABC 4. (2) (A′ + B + C′)(A + B′ + C) (4) (A′ + B)(B′ + C)(A′ + C) (6) AB′C′ Note that it is possible for a minterm or maxterm expansion to have only one term. (a) Given a minterm in terms of its variables, the procedure for conversion to decimal notation is (1) Replace each complemented variable with a _____ and replace each uncomplemented variable with a _____. (2) Convert the resulting binary number to decimal. (b) Convert the minterm AB′C′DE to decimal notation. (c) Given that m13 is a minterm of the variables A, B, C, D, and E, write the minterm in terms of these variables. (d) Given a maxterm in terms of its variables, the procedure for conversion to decimal notation is (1) Replace each complemented variable with a _____ and replace each uncomplemented variable with a _____. (2) Group these 0's and 1's to form a binary number and convert to decimal. (e) Convert the maxterm A′ + B + C + D′ + E′ to decimal notation. (f) Given that M13 is a maxterm of the variables A, B, C, D, and E, write the maxterm in terms of these variables. (g) Check your answers to (b), (c), (e), and (f) by using the relation Mi = mi′. (h) Given f(a, b, c, d, e) = Π M(0, 10, 28), express f in terms of a, b, c, d, and e. (Your answer should contain only five complemented variables.) 91 Applications of Boolean Algebra Minterm and Maxterm Expansions 5. Study Section 4.4, General Minterm and Maxterm Expansions. Make sure that you understand the notation here and can follow the algebra in all of the equations. If you have difficulty with this section, ask for help before you take the readiness test. (a) How many different switching functions of four variables are possible? n (b) Explain why there are 22 switching functions of n variables. (c) Write the function of Figure 4-1 in the form of Equation (4-13) and show that it reduces to Equation (4-3). (d) For Equation (4-19), write out the indicated summations in full for the case n = 2. (e) Study Tables 4-3 and 4-4 carefully and make sure you understand why each table entry is valid. Use the truth table for f and f′ (Figure 4-1) to verify the entries in Table 4-4. If you understand the relationship between Table 4-3 and the truth table for f and f′, you should be able to perform the conversions without having to memorize the table. (f) Given that f (A, B, C) = Σm (0, 1, 3, 4, 7) The maxterm expansion for f is ______________________ The minterm expansion for f′ is ______________________ The maxterm expansion for f′ is ______________________ (g) Work Problems 4.3 and 4.4. 6. Study Section 4.5, Incompletely Specified Functions. (a) State two reasons why some functions have don't-care terms. (b) Given the following table, write the minterm expansion for Z in decimal form. (c) Write the maxterm expansion in decimal form. (d) Work Problems 4.5 and 4.6. A 0 0 0 0 1 1 1 1 B 0 0 1 1 0 0 1 1 C 0 1 0 1 0 1 0 1 Z 1 X 0 X X 1 0 0 92 Unit 4 7. Study Section 4.6, Examples of Truth Table Construction. Finding the truth table from the problem statement is probably the most difficult part of the process of designing a switching circuit. Make sure that you understand how to do this. 8. Work Problems 4.7 through 4.10. 9. Study Section 4.7, Design of Binary Adders. (a) For the given parallel adder, show the 0's and 1's at the full adder (FA) inputs and outputs when the following unsigned numbers are added: 11 + 14 = 25. Verify that the result is correct if C4S3S2S1S0 is taken as a 5-bit sum. If the sum is limited to 4 bits, explain why this is an overflow condition. S3 C4 FA S2 FA S1 FA S0 FA C0 (b) Review Section 1.4, Representation of Negative Numbers. If we use the 2's complement number system to add (−5) + (−2), verify that the FA inputs and outputs are exactly the same as in part (a). However, for 2's complement, the interpretation of the results is quite different. After discarding C4, verify that the resultant 4-bit sum is correct, and therefore no overflow has occurred. (c) If we use the 1's complement number system to add (−5) + (−2), show the FA inputs and outputs on the diagram below before the end-around carry is added in. Assume that C0 is initially 0. Then add the end-around carry (C4) to the rightmost FA, add the new carry (C1) into the next cell, and continue until no further changes occur. Verify that the resulting sum is the correct 1's complement representation of −7. C4 10. (a) FA FA FA FA C0 Work the following subtraction example. As you subtract each column, place a 1 over the next column if you have to borrow, otherwise place a 0. For each column, as you compute xi − yi − bi, fill in the corresponding values of bi+1 and di in the truth table. If you have done this correctly, the resulting truth table should match the full subtracter truth table (Table 4-6). Applications of Boolean Algebra Minterm and Maxterm Expansions ← borrows ←X ←Y ← difference 11000110 −0 1 0 1 1 0 1 0 xi 0 0 0 0 1 1 1 1 yi 0 0 1 1 0 0 1 1 bi 0 1 0 1 0 1 0 1 93 bi+1 di (b) Work Problems 4.11 and 4.12. 11. Read the following and then work Problem 4.13 or 4.14 as assigned: When looking at an expression to determine the required number of gates, keep in mind that the number of required gates is generally not equal to the number of AND and OR operations which appear in the expression. For example, AB + CD + EF(G + H) contains four AND operations and three OR operations, but it only requires three AND gates and two OR gates: A B C D G E F H 12. Reread the objectives of this unit. Make sure that you understand the difference in the procedures for converting maxterms and minterms from decimal to algebraic notation. When you are satisfied that you can meet the objectives, take the readiness test. When you come to take the readiness test, turn in a copy of your solution to assigned simulation exercise. Applications of Boolean Algebra Minterm and Maxterm Expansions In this unit you will learn how to design a combinational logic circuit starting with a word description of the desired circuit behavior. The first step is usually to translate the word description into a truth table or into an algebraic expression. Given the truth table for a Boolean function, two standard algebraic forms of the function can be derived—the standard sum of products (minterm expansion) and the standard product of sums (maxterm expansion). Simplification of either of these standard forms leads directly to a realization of the circuit using AND and OR gates. 4.1 Conversion of English Sentences to Boolean Equations The three main steps in designing a single-output combinational switching circuit are 1. 2. 3. Find a switching function that specifies the desired behavior of the circuit. Find a simplified algebraic expression for the function. Realize the simplified function using available logic elements. For simple problems, it may be possible to go directly from a word description of the desired behavior of the circuit to an algebraic expression for the output function. In other cases, it is better to first specify the function by means of a truth table and then derive an algebraic expression from the truth table. Logic design problems are often stated in terms of one or more English sentences. The first step in designing a logic circuit is to translate these sentences into Boolean equations. In order to do this, we must break down each sentence into phrases and associate a Boolean variable with each phrase. If a phrase can have a value of true or false, then we can represent that phrase by a Boolean variable. Phrases such as “she goes to the store” or “today is Monday” can be either true or false, but a command like “go to the store” has no truth value. If a sentence has several phrases, we will mark each phrase with a brace. The following sentence has three phrases: Mary watches TV if it is Monday night and she has finished her homework. 94 Applications of Boolean Algebra Minterm and Maxterm Expansions 95 The “if” and “and” are not included in any phrase; they show the relationships among the phrases. We will define a two-valued variable to indicate the truth or falsity of each phrase: F = 1 if “Mary watches TV” is true; otherwise, F = 0. A = 1 if “it is Monday night” is true; otherwise, A = 0. B = 1 if “she has finished her homework” is true; otherwise B = 0. Because F is “true” if A and B are both “true”, we can represent the sentence by F = A·B The following example illustrates how to go from a word statement of a problem directly to an algebraic expression which represents the desired circuit behavior. An alarm circuit is to be designed which operates as follows: The alarm will ring iff the alarm switch is turned on and the door is not closed, or it is after 6 p.m. and the window is not closed. The first step in writing an algebraic expression which corresponds to the above sentence is to associate a Boolean variable with each phrase in the sentence. This variable will have a value of 1 when the phrase is true and 0 when it is false. We will use the following assignment of variables: The alarm will ring iff the alarm switch is on Z and A the door is not closed or it is after 6 P.M. B′ the window is not closed. and C D′ This assignment implies that if Z = 1, the alarm will ring. If the alarm switch is turned on, A = 1, and if it is after 6 p.m., C = 1. If we use the variable B to represent the phrase “the door is closed”, then B′ represents “the door is not closed”. Thus, B = 1 if the door is closed, and B′ = 1(B = 0) if the door is not closed. Similarly, D = 1 if the window is closed, and D′ = 1 if the window is not closed. Using this assignment of variables, the above sentence can be translated into the following Boolean equation: Z = AB′ + CD′ This equation corresponds to the following circuit: A B Z C D In this circuit, A is a signal which is 1 when the alarm switch is on, C is a signal from a time clock which is 1 when it is after 6 p.m., B is a signal from a switch on the door 96 Unit 4 which is 1 when the door is closed, and similarly D is 1 when the window is closed. The output Z is connected to the alarm so that it will ring when Z = 1. 4.2 Combinational Logic Design Using a Truth Table The next example illustrates logic design using a truth table. A switching circuit has three inputs and one output, as shown in Figure 4-1(a). The inputs A, B, and C represent the first, second, and third bits, respectively, of a binary number N. The output of the circuit is to be f = 1 if N ≥ 0112 and f = 0 if N < 0112. The truth table for f is shown in Figure 4-1(b). FIGURE 4-1 Combinational Circuit with Truth Table © Cengage Learning 2014 A f B C (a) A B C f f′ 0 0 0 0 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 0 0 1 1 0 1 0 1 0 1 0 1 (b) Next, we will derive an algebraic expression for f from the truth table by using the combinations of values of A, B, and C for which f = 1. The term A′BC is 1 only if A = 0, B = 1, and C = 1. Similarly, the term AB′C′ is 1 only for the combination 100, AB′C is 1 only for 101, ABC′ is 1 only for 110, and ABC is 1 only for 111. ORing these terms together yields f = A′BC + AB′C′ + AB′C + ABC′ + ABC (4-1) This expression equals 1 if A, B, and C take on any of the five combinations of values 011, 100, 101, 110, or 111. If any other combination of values occurs, f is 0 because all five terms are 0. Equation (4-1) can be simplified by first combining terms using the uniting theorem and then eliminating A′ using the elimination theorem: f = A′BC + AB′ + AB = A′BC + A = A + BC Equation (4-2) leads directly to the following circuit: B C A f (4-2) Applications of Boolean Algebra Minterm and Maxterm Expansions 97 Instead of writing f in terms of the 1's of the function, we may also write f in terms of the 0's of the function. The function defined by Figure 4-1 is 0 for three combinations of input values. Observe that the term A + B + C is 0 only if A = B = C = 0. Similarly, A + B + C′ is 0 only for the input combination 001, and A + B′ + C is 0 only for the combination 010. ANDing these terms together yields f = (A + B + C)(A + B + C′)(A + B′ + C) (4-3) This expression equals 0 if A, B, and C take on any of the combinations of values 000, 001, or 010. For any other combination of values, f is 1 because all three terms are l. Because Equation (4-3) represents the same function as Equation (4-1) they must both reduce to the same expression. Combining terms and using the second distributive law, Equation (4-3) simplifies to f = (A + B)(A + B′ + C) = A + B(B′ + C) = A + BC (4-4) which is the same as Equation (4-2). Another way to derive Equation (4-3) is to first write f′ as a sum of products, and then complement the result. From Figure 4-1, f′is 1 for input combinations ABC = 000, 001, and 010, so f′ = A′B′C′ + A′B′C + A′BC′ Taking the complement of f′ yields Equation (4-3). 4.3 Minterm and Maxterm Expansions Each of the terms in Equation (4-1) is referred to as a minterm. In general, a minterm of n variables is a product of n literals in which each variable appears exactly once in either true or complemented form, but not both. (A literal is a variable or its complement.) Table 4-1 lists all of the minterms of the three variables A, B, and C. Each minterm has a value of 1 for exactly one combination of values of the variables A, B, and C. Thus if A = B = C = 0, A′B′C′ = 1; if A = B = 0 and C = 1, A′B′C = 1; and so forth. Minterms are often written in abbreviated form—A′B′C′ is designated m0, A′B′C is designated m1, etc. In general, the minterm which corresponds to row i of the truth table is designated mi (i is usually written in decimal). TABLE 4-1 Minterms and Maxterms for Three Variables © Cengage Learning 2014 Row No. 0 1 2 3 4 5 6 7 A 0 0 0 0 1 1 1 1 B 0 0 1 1 0 0 1 1 C 0 1 0 1 0 1 0 1 Minterms A′B′C′ = m0 A′B′C = m1 A′BC′ = m2 A′BC = m3 AB′C′ = m4 AB′C = m5 ABC′ = m6 ABC = m7 Maxterms A + B + C = M0 A + B + C′ = M1 A + B′ + C = M2 A + B′ + C′ = M3 A′ + B + C = M4 A′ + B + C′ = M5 A′ + B′ + C = M6 A′ + B′ + C′ = M7 98 Unit 4 When a function f is written as a sum of minterms as in Equation (4-1), this is referred to as a minterm expansion or a standard sum of products.1 If f = 1 for row i of the truth table, then mi must be present in the minterm expansion because mi = 1 only for the combination of values of the variables corresponding to row i of the table. Because the minterms present in f are in one-to-one correspondence with the 1's of f in the truth table, the minterm expansion for a function f is unique. Equation (4-1) can be rewritten in terms of m-notation as f(A, B, C) = m3 + m4 + m5 + m6 + m7 (4-5) This can be further abbreviated by listing only the decimal subscripts in the form f(A, B, C) = Σ m(3, 4, 5, 6, 7) (4-5a) Each of the sum terms (or factors) in Equation (4-3) is referred to as a maxterm. In general, a maxterm of n variables is a sum of n literals in which each variable appears exactly once in either true or complemented form, but not both. Table 4-1 lists all of the maxterms of the three variables A, B, and C. Each maxterm has a value of 0 for exactly one combination of values for A, B, and C. Thus, if A = B = C = 0, A + B + C = 0; if A = B = 0 and C = 1, A + B + C′ = 0; and so forth. Maxterms are often written in abbreviated form using M-notation. The maxterm which corresponds to row i of the truth table is designated Mi. Note that each maxterm is the complement of the corresponding minterm, that is, Mi = m′.i When a function f is written as a product of maxterms, as in Equation (4-3), this is referred to as a maxterm expansion or standard product of sums. If f = 0 for row i of the truth table, then Mi must be present in the maxterm expansion because Mi = 0 only for the combination of values of the variables corresponding to row i of the table. Note that the maxterms are multiplied together so that if any one of them is 0, f will be 0. Because the maxterms are in one-to-one correspondence with the 0's of f in the truth table, the maxterm expansion for a function f is unique. Equation (4-3) can be rewritten in M-notation as f(A, B, C) = M0 M1M2 (4-6) This can be further abbreviated by listing only the decimal subscripts in the form f(A, B, C) = Π M(0, 1, 2) (4-6a) where Π means a product. Because if f ≠ 1 then f = 0, it follows that if mi is not present in the minterm expansion of f , then Mi is present in the maxterm expansion. Thus, given a minterm expansion of an n-variable function f in decimal notation, the maxterm expansion is obtained by listing those decimal integers (0 ≤ i ≤ 2n − 1) not in the minterm list. Using this method, Equation (4-6a) can be obtained directly from Equation (4-5a). 1 Other names used in the literature for standard sum of products are canonical sum of products and disjunctive normal form. Similarly, a standard product of sums may be called a canonical product of sums or a conjunctive normal form. Applications of Boolean Algebra Minterm and Maxterm Expansions 99 Given the minterm or maxterm expansions for f , the minterm or maxterm expansions for the complement of f are easy to obtain. Because f′ is 1 when f is 0, the minterm expansion for f′ contains those minterms not present in f . Thus, from Equation (4-5), f′ = m0 + m1 + m2 = Σ m(0, 1, 2) (4-7) Similarly, the maxterm expansion for f′ contains those maxterms not present in f . From Equation (4-6), f′ = Π M(3, 4, 5, 6, 7) = M3M4M5M6M7 (4-8) Because the complement of a minterm is the corresponding maxterm, Equation (4-8) can be obtained by complementing Equation (4-5): f′ = (m3 + m4 + m5 + m6 + m7)′ = m′3 m′4 m′5 m′6 m′7 = M3M4M5M6M7 Similarly, Equation (4-7) can be obtained by complementing Equation (4-6): f′ = (M0M1M2)′ = M0′ + M1′ + M2′ = m0 + m1 + m2 A general switching expression can be converted to a minterm or maxterm expansion either using a truth table or algebraically. If a truth table is constructed by evaluating the expression for all different combinations of the values of the variables, the minterm and maxterm expansions can be obtained from the truth table by the methods just discussed. Another way to obtain the minterm expansion is to first write the expression as a sum of products and then introduce the missing variables in each term by applying the theorem X + X′ = 1. Example Find the minterm expansion of f(a, b, c, d) = a′(b′ + d) + acd′. f = a′b′ + a′d + acd′ = a′b′(c + c′)(d + d′) + a′d(b + b′)(c + c′) + acd′(b + b′) = a′b′c′d′ + a′b′c′d + a′b′cd′ + a′b′cd + a′b′c′d + a′b′cd + a′bc′d + a′bcd + abcd′ + ab′cd′ (4-9) Duplicate terms have been crossed out, because X + X = X. This expression can then be converted to decimal notation: f = a′b′c′d′ + a′b′c′d + a′b′cd′ + a′b′cd + a′bc′d + a′bcd + abcd′ + ab′cd′ 0 0 0 0 0 0 0 1 0 0 1 0 0 0 11 0 10 1 0 111 1 1 1 0 10 10 f = Σ m(0, 1, 2, 3, 5, 7, 10, 14) (4-10) The maxterm expansion for f can then be obtained by listing the decimal integers (in the range 0 to 15) which do not correspond to minterms of f : f = Π M(4, 6, 8, 9, 11, 12, 13, 15) 100 Unit 4 An alternate way of finding the maxterm expansion is to factor f to obtain a product of sums, introduce the missing variables in each sum term by using XX′ = 0, and then factor again to obtain the maxterms. For Equation (4-9), f = = = = a′(b′ + d) + acd′ (a′ + cd′)(a + b′ + d) = (a′ + c)(a′ + d′)(a + b′ + d) (a′ + bb′ + cc′ + dd′)(a′ + bb′ + cc′ + d′)(a + b′ + cc′ + d) (a′ + bb′ + c + d)(a′ + bb′ + c + d′)(a′ + bb′ + c′ + d′)(a + b′ + cc′ + d) = (a′ + b + c + d)(a′ + b′ + c + d)(a′ + b + c + d′)(a′ + b′ + c + d′) 1000 1100 1001 1101 (a′ + b + c′ + d′)(a′ + b′ + c′ + d′)(a + b′ + c + d)(a + b′ + c′ + d) 1011 1111 0100 0110 = Π M(4, 6, 8, 9, 11, 12, 13, 15) (4-11) Note that when translating the maxterms to decimal notation, a primed variable is first replaced with a 1 and an unprimed variable with a 0. Because the terms in the minterm expansion of a function F correspond one-toone with the rows of the truth table for which F = 1, the minterm expansion of F is unique. Thus, we can prove that an equation is valid by finding the minterm expansion of each side and showing that these expansions are the same. Example Show that a′c + b′c′ + ab = a′b′ + bc + ac′. We will find the minterm expansion of each side by supplying the missing variables. For the left side, a′c(b + b′) + b′c′(a + a′) + ab(c + c′) = a′bc + a′b′c + ab′c′ + a′b′c′ + abc + abc′ = m3 + m1 + m4 + m0 + m7 + m6 For the right side, a′b′(c + c′) + bc(a + a′) + ac′(b + b′) = a′b′c + a′b′c + abc + a′bc + abc′ + ab′c′ = m1 + m0 + m7 + m3 + m6 + m4 Because the two minterm expansions are the same, the equation is valid. 4.4 General Minterm and Maxterm Expansions Table 4-2 represents a truth table for a general function of three variables. Each ai is a constant with a value of 0 or 1. To completely specify a function, we must assign values to all of the ai's. Because each ai can be specified in two ways, there are 28 Applications of Boolean Algebra Minterm and Maxterm Expansions TABLE 4-2 General Truth Table for Three Variables © Cengage Learning 2014 A 0 0 0 0 1 1 1 1 B 0 0 1 1 0 0 1 1 C 0 1 0 1 0 1 0 1 101 F a0 a1 a2 a3 a4 a5 a6 a7 ways of filling the F column of the truth table; therefore, there are 256 different functions of three variables (this includes the degenerate cases, F identically equal to 0 and F identically equal to 1). For a function of n variables, there are 2n rows in the n truth table, and because the value of F can be 0 or 1 for each row, there are 22 possible functions of n variables. From Table 4-2, we can write the minterm expansion for a general function of three variables as follows: 7 F = a0m0 + a1m1 + a2m2 + · · · + a7m7 = a aimi (4-12) i=0 Note that if ai = 1, minterm mi is present in the expansion; if ai = 0, the corresponding minterm is not present. The maxterm expansion for a general function of three variables is 7 F = (a0 + M0)(a1 + M1)(a2 + M2) · · · (a7 + M7) = q (ai + Mi) (4-13) i=0 Note that if ai = 1, ai + Mi = 1, and Mi drops out of the expansion; however, Mi is present if ai = 0. From Equation (4-13), the minterm expansion of F′ is 7 7 7 ′ F′ = c q (ai + Mi) d = a ai′Mi′ = a ai′mi i=0 i=0 (4-14) i=0 Note that all minterms which are not present in F are present in F′. From Equation (4-12), the maxterm expansion of F′ is 7 7 7 ′ F′ = c a aimi d = q (ai′ + mi′) = q (ai′ + Mi) i=0 i=0 (4-15) i=0 Note that all maxterms which are not present in F are present in F′. Generalizing Equations (4-12), (4-13), (4-14), and (4-15) to n variables, we have 2n −1 2n −1 i=0 i=0 F = a aimi = q (ai + Mi) (4-16) 102 Unit 4 2n −1 2n −1 F′ = a ai′ mi = q (ai′ + Mi) (4-17) i=0 i=0 Given two different minterms of n variables, mi and mj, at least one variable appears complemented in one of the minterms and uncomplemented in the other. Therefore, if i ≠ j, mi mj = 0. For example, for n = 3, m1m3 = (A′B′C)(A′BC) = 0. Given minterm expansions for two functions 2n −1 2n −1 f1 = a ai mi f2 = a bj mj i=0 j=0 (4-18) the product is 2n −1 2n −1 2n −1 2n −1 i=0 j=0 i=0 j=0 f1 f2 = a a ai mi b a a bj mj b = a a ai bj mi mj 2 −1 n = a ai bi mi (because mi mj = 0 unless i = j) (4-19) i=0 Note that all of the cross-product terms (i ≠ j) drop out so that f1 f2 contains only those minterms which are present in both f1 and f2. For example, if f1 = Σ m(0, 2, 3, 5, 9, 11) and f2 = Σ m(0, 3, 9, 11, 13, 14) f1 f2 = Σ m(0, 3, 9, 11) Table 4-3 summarizes the procedures for conversion between minterm and maxterm expansions of F and F′, assuming that all expansions are written as lists of decimal numbers. When using this table, keep in mind that the truth table for an n-variable function has 2n rows so that the minterm (or maxterm) numbers range from 0 to 2n − 1. Table 4-4 illustrates the application of Table 4-3 to the threevariable function given in Figure 4-1. TABLE 4-3 Conversion of Forms Minterm Expansion of F GIVEN FORM © Cengage Learning 2014 DESIRED FORM Maxterm Expansion of F Minterm Expansion of F′ Maxterm Expansion of F′ Minterm Expansion of F ____________ maxterm nos. are those nos. not on the minterm list for F list minterms not present in F maxterm nos. are the same as minterm nos. of F Maxterm Expansion of F minterm nos. are those nos. not on the maxterm list for F ____________ minterm nos. are the same as maxterm nos. of F list maxterms not present in F TABLE 4-4 Application of Table 4.3 © Cengage Learning 2014 GIVEN FORM Applications of Boolean Algebra Minterm and Maxterm Expansions DESIRED FORM Maxterm Minterm Expansion Expansion of f of f f= Σ m(3, 4, 5, 6, 7) f= Π M(0, 1, 2) ____________ Σ m(3, 4, 5, 6, 7) Minterm Expansion of f ′ Π M(0, 1, 2) Σ m(0, 1, 2) _________ Σ m(0, 1, 2) 103 Maxterm Expansion of f ′ Π M(3, 4, 5, 6, 7) Π M(3, 4, 5, 6, 7) 4.5 Incompletely Specified Functions A large digital system is usually divided into many subcircuits. Consider the following example in which the output of circuit N1 drives the input of circuit N2. w x y z A N1 B N2 F C Let us assume that the output of N1 does not generate all possible combinations of values for A, B, and C. In particular, we will assume that there are no combinations of values for w, x, y, and z which cause A, B, and C to assume values of 001 or 110. Hence, when we design N2, it is not necessary to specify values of F for ABC = 001 or 110 because these combinations of values can never occur as inputs to N2. For example, F might be specified by Table 4-5. The X's in the table indicate that we don't care whether the value of 0 or 1 is assigned to F for the combinations ABC = 001 or 110. In this example, we don't care what the value of F is because these input combinations never occur anyway. The function F is then incompletely specified. The minterms A′B′C and ABC′ are referred to as don't-care minterms, since we don't care whether they are present in the function or not. TABLE 4-5 Truth Table with Don't-Cares © Cengage Learning 2014 A 0 0 0 0 1 1 1 1 B 0 0 1 1 0 0 1 1 C 0 1 0 1 0 1 0 1 F 1 X 0 1 0 0 X 1 104 Unit 4 When we realize the function, we must specify values for the don't-cares. It is desirable to choose values which will help simplify the function. If we assign the value 0 to both X's, then F = A′B′C′ + A′BC + ABC = A′B′C′ + BC If we assign 1 to the first X and 0 to the second, then F = A′B′C′ + A′B′C + A′BC + ABC = A′B′ + BC If we assign 1 to both X's, then F = A′B′C′ + A′B′C + A′BC + ABC′ + ABC = A′B′ + BC + AB The second choice of values leads to the simplest solution. We have seen one way in which incompletely specified functions can arise, and there are many other ways. In the preceding example, don't-cares were present because certain combinations of circuit inputs did not occur. In other cases, all input combinations may occur, but the circuit output is used in such a way that we do not care whether it is 0 or 1 for certain input combinations. When writing the minterm expansion for an incompletely specified function, we will use m to denote the required minterms and d to denote the don't-care minterms. Using this notation, the minterm expansion for Table 4-5 is F = Σ m(0, 3, 7) + Σ d(1, 6) For each don't-care minterm there is a corresponding don't-care maxterm. For example, if F = X (don't-care) for input combination 001, m1 is a don't-care minterm and M1 is a don't-care maxterm. We will use D to represent a don't-care maxterm, and we write the maxterm expansion of the function in Table 4-5 as F = Π M(2, 4, 5) · Π D(1, 6) which implies that maxterms M2, M4, and M5 are present in F and don't-care maxterms M1 and M6 are optional. 4.6 Examples of Truth Table Construction Example 1 We will design a simple binary adder that adds two 1-bit binary numbers, a and b, to give a 2-bit sum. The numeric values for the adder inputs and output are as follows: a b Sum 0 0 1 1 00 01 01 10 0 1 0 1 (0 + 0 = 0) (0 + 1 = 1) (1 + 0 = 1) (1 + 1 = 2) Applications of Boolean Algebra Minterm and Maxterm Expansions 105 We will represent inputs to the adder by the logic variables A and B and the 2-bit sum by the logic variables X and Y, and we construct a truth table: A 0 0 1 1 B 0 1 0 1 X 0 0 0 1 Y 0 1 1 0 Because a numeric value of 0 is represented by a logic 0 and a numeric value of 1 by a logic l, the 0's and 1's in the truth table are exactly the same as in the previous table. From the truth table, X = AB and Y = A′B + AB′ = A ⊕ B Example 2 An adder is to be designed which adds two 2-bit binary numbers to give a 3-bit binary sum. Find the truth table for the circuit. The circuit has four inputs and three outputs as shown: N1 N2 A B C D X Y Z N3 N1 $%& AB 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 TRUTH TABLE: N2 N $%& $'%3'& XYZ CD 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1 1 1 0 0 1 1 0 1 1 0 1 1 0 0 1 0 0 1 0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0 Inputs A and B taken together represent a binary number N1. Inputs C and D taken together represent a binary number N2. Outputs X, Y, and Z taken together represent a binary number N3, where N3 = N1 + N2 (+ of course represents ordinary addition here). In this example we have used A, B, C, and D to represent both numeric values and logic values, but this should not cause any confusion because the numeric and 106 Unit 4 logic values are the same. In forming the truth table, the variables were treated like binary numbers having numeric values. Now we wish to derive the switching functions for the output variables. In doing so, we will treat A, B, C, D, X, Y, and Z as switching variables having nonnumeric values 0 and 1. (Remember that in this case the 0 and 1 may represent low and high voltages, open and closed switches, etc.) From inspection of the table, the output functions are X(A, B, C, D) = Σ m(7, 10, 11, 13, 14, 15) Y(A, B, C, D) = Σ m(2, 3, 5, 6, 8, 9, 12, 15) Z(A, B, C, D) = Σ m(1, 3, 4, 6, 9, 11, 12, 14) Example 3 Design an error detector for 6-3-1-1 binary-coded-decimal digits. The output (F) is to be 1 iff the four inputs (A, B, C, D) represent an invalid code combination. The valid 6-3-1-1 code combinations are listed in Table 1-2. If any other combination occurs, this is not a valid 6-3-1-1 binary-coded-decimal digit, and the circuit output should be F = 1 to indicate that an error has occurred. This leads to the following truth table: A 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 B 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 C D 0 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 F 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 1 The corresponding output function is F = Σ m(2, 6, 10, 13, 14, 15) = A′B′CD′ + A′BCD′ + AB′CD′ +' ABCD′ + ABC′D + ABCD (''''* '((''''* ('''' (= A′CD′ + ACD′ + ABD = CD′ + ABD (''''* Applications of Boolean Algebra Minterm and Maxterm Expansions 107 The realization using AND and OR gates is C F D′ A B D Example 4 The four inputs to a circuit (A, B, C, D) represent an 8-4-2-1 binary-coded-decimal digit. Design the circuit so that the output (Z) is 1 iff the decimal number represented by the inputs is exactly divisible by 3. Assume that only valid BCD digits occur as inputs. The digits 0, 3, 6, and 9 are exactly divisible by 3, so Z = 1 for the input combinations ABCD = 0000, 0011, 0110, and 1001. The input combinations 1010, 1011, 1100, 1101, 1110, and 1111 do not represent valid BCD digits and will never occur, so Z is a don't-care for these combinations. This leads to the following truth table: A 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 B C 0 0 0 0 0 1 0 1 1 0 1 0 1 1 1 1 0 0 0 0 0 1 0 1 1 0 1 0 1 1 1 1 D 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 Z 1 0 0 1 0 0 1 0 0 1 X X X X X X The corresponding output function is Z = Σ m(0, 3, 6, 9) + Σ d(10, 11, 12, 13, 14, 15) In order to find the simplest circuit which will realize Z, we must choose some of the don't-cares (X's) to be 0 and some to be 1. The easiest way to do this is to use a Karnaugh map as described in Unit 5. 108 Unit 4 4.7 Design of Binary Adders and Subtracters In this section, we will design a parallel adder that adds two 4-bit unsigned binary numbers and a carry input to give a 4-bit sum and a carry output (see Figure 4-2). One approach would be to construct a truth table with nine inputs and five outputs and then derive and simplify the five output equations. Because each equation would be a function of nine variables before simplification, this approach would be very difficult, and the resulting logic circuit would be very complex. A better method is to design a logic module that adds two bits and a carry, and then connect four of these modules together to form a 4-bit adder as shown in Figure 4-3. Each of the modules is called a full adder. The carry output from the first full adder serves as the carry input to the second full adder, etc. FIGURE 4-2 Parallel Adder for 4-Bit Binary Numbers S3 S2 S0 4-bit Parallel Adder C4 © Cengage Learning 2014 S1 C0 A3 B3 A2 B2 A1 B1 A0 B0 FIGURE 4-3 Parallel Adder Composed of Four Full Adders C4 © Cengage Learning 2014 1 S3 0 S2 C3 Full Adder A3 0 B3 1 1 1 S1 C2 Full Adder A2 1 B2 0 0 1 S0 C1 Full Adder A1 1 B1 1 1 0 C0 Full Adder A0 0 B0 1 1 end-around carry for 1's complement In the example of Figure 4-3, we perform the following addition: 10110 (carries) 1011 + 1011 10110 The full adder to the far right adds A0 + B0 + C0 = 1 + 1 + 0 to give a sum of 102, which gives a sum S0 = 0 and a carry out of C1 = 1. The next full adder adds A1 + B1 + C1 = 1 + 1 + 1 = 112, which gives a sum S1 = 1 and a carry C2 = 1. The carry continues to propagate from right to left until the left cell produces a final carry of C4 = 1. Applications of Boolean Algebra Minterm and Maxterm Expansions FIGURE 4-4 Truth Table for a Full Adder X Y Cin Cout Full Adder Sum © Cengage Learning 2014 X Y Cin Cout Sum 0 0 0 0 1 1 1 1 0 0 1 1 0 0 1 1 0 1 0 1 0 1 0 1 0 0 0 1 0 1 1 1 0 1 1 0 1 0 0 1 109 Figure 4-4 gives the truth table for a full adder with inputs X, Y, and Cin. The outputs for each row of the table are found by adding up the input bits (X + Y + Cin) and splitting the result into a carry out (Ci+1) and a sum bit (Si) . For example, in the 101 row 1 + 0 + 1 = 102, so Ci+1 = 1 and Si = 0. Figure 4-5 shows the implementation of the full adder using gates. The logic equations for the full adder derived from the truth table are Sum = X′Y′Cin + X′YC′in + XY′C′in + XYCin = X′(Y′Cin + YC′in) + X(Y′C′in + YCin) = X′(Y ⊕ Cin) + X(Y ⊕ Cin)′ = X ⊕ Y ⊕ Cin (4-20) Cout = X′YCin + XY′Cin + XYC′in + XYCin = (X′YCin + XYCin) + (XY′Cin + XYCin) + (XYC′in + XYCin) = YCin + XCin + XY (4-21) Note that the term XYCin was used three times in simplifying Cout. Figure 4-5 shows the logic circuit for Equations (4-20) and (4-21). FIGURE 4-5 Implementation of Full Adder © Cengage Learning 2014 x y x y cin Sum x cin cout y cin Although designed for unsigned binary numbers, the parallel adder of Figure 4-3 can also be used for signed binary numbers with negative numbers expressed in complement form. When 2's complement is used, the last carry (C4) is discarded, and there is no carry into the first cell. Because C0 = 0, the equations for the first cell may be simplified to S0 = A0 ⊕ B0 and C1 = A0 B0 When 1's complement is used, the end-around carry is accomplished by connecting C4 to the C0 input, as shown by the dashed line in Figure 4-3. When adding signed binary numbers with negative numbers expressed in complement form, the sign bit of the sum is wrong when an overflow occurs. That is, an overflow has occurred if adding two positive numbers gives a negative result, or adding two negative numbers gives a positive result. We will define a signal V that 110 Unit 4 is 1 when an overflow occurs. For Figure 4-3, we can use the sign bits of A, B, and S (the sum) to determine the value of V: V = A3′ B3′ S3 + A3B3S3′ (4-22) If the number of bits is large, a parallel binary adder of the type shown in Figure 4-4 may be rather slow because the carry generated in the first cell might have to propagate all of the way to the last cell. Subtraction of binary numbers is most easily accomplished by adding the complement of the number to be subtracted. To compute A − B, add the complement of B to A. This gives the correct answer because A + (−B) = A − B. Either 1's or 2's complement is used depending on the type of adder employed. The circuit of Figure 4-6 may be used to form A − B using the 2's complement representation for negative numbers. The 2's complement of B can be formed by first finding the 1's complement and then adding 1. The 1's complement is formed by inverting each bit of B, and the addition of 1 is effectively accomplished by putting a 1 into the carry input of the first full adder. S4 FIGURE 4-6 Binary Subtracter Using Full Adders © Cengage Learning 2014 c5 (Ignore last carry) Full Adder B′4 A4 Example S3 B4 c4 S2 Full Adder c3 c2 Full Adder B′3 A3 B3 S1 Full Adder B′2 A2 B2 c1 = 1 B′1 A1 B1 A = 0110 (+6) B = 0011 (+3) The adder output is 0110 +1100 + 1 (1) 0011 = 3 = (+6) (1's complement of 3) (first carry input) 6−3 Alternatively, direct subtraction can be accomplished by employing a full subtracter in a manner analogous to a full adder. A block diagram for a parallel subtracter which subtracts Y from X is shown in Figure 4-7. The first two bits are subtracted in the rightmost cell to give a difference d1, and a borrow signal (b2 = 1) is generated if it is necessary to borrow from the next column. A typical cell (cell i) has inputs xi, yi, and bi, and outputs bi+1 and di. An input bi = 1 indicates that we must borrow 1 from xi in that cell, and borrowing 1 from xi is equivalent to subtracting 1 from xi. In cell i, Applications of Boolean Algebra Minterm and Maxterm Expansions dn FIGURE 4-7 Parallel Subtracter © Cengage Learning 2014 bn + 1 di Full Subtracter bn bi + 1 d1 d2 Full Subtracter bi b3 111 Full Subtracter b2 Full Subtracter b1 = 0 Cell i xn TABLE 4-6 Truth Table for Binary Full Subtracter © Cengage Learning 2014 xi 0 0 0 0 1 1 1 1 yi 0 0 1 1 0 0 1 1 yn xi yi x2 y2 x1 y1 bi bi+1di 0 0 0 1 1 1 0 1 1 1 1 0 0 0 1 1 1 0 0 0 0 0 1 1 1 bits and bi and yi are subtracted from xi to form the difference di, and a borrow signal (bi+1 = 1) is generated if it is necessary to borrow from the next column. Table 4-6 gives the truth table for a binary full subtracter. Consider the following case, where xi = 0, yi = 1 and bi = 1: xi −bi −yi di Column i Before Borrow 0 −1 −1 Column i After Borrow 10 −1 −1 0 (bi+1 = 1) Note that in column i, we cannot immediately subtract yi and bi from xi. Hence, we must borrow from column i + 1. Borrowing 1 from column i + 1 is equivalent to setting bi+1 to 1 and adding 10 (210) to xi. We then have di = 10 − 1 − 1 = 0. Verify that Table 4-6 is correct for the other input combinations and use it to work out several examples of binary subtraction. The ripple carry adder is relatively slow because, in the worst case, a carry propagates through all stages of the adder, and there are two gate delays per stage. There are several techniques for reducing the carry propagation time. One is called a carrylookahead adder. In the parallel adder the carry out of the ith stage can be written as Ci+1 = Ai Bi + Ci(Ai + Bi) = Ai Bi + Ci(Ai ⊕ Bi) = Gi + PiCi 112 Unit 4 where Gi = Ai Bi indicates the condition for the ith stage to generate a carry out and Pi = Ai ⊕ Bi (or Pi = Ai + Bi) indicates the condition for the ith stage to propagate a carry in to the carry out. Then Ci+2 can be expressed in terms of Ci. Ci+2 = Gi+1 + Pi+1Ci+1 = Gi+1 + (Gi + Ci Pi)Pi+1 = Gi+1 + Pi+1Gi + Pi+1PiCi This can be continued to express Ci+2, Ci+3, etc. in terms of Ci. Ci+1 Ci+2 Ci+3 Ci+4 = Gi + PiCi = Gi+1 + Pi+1Gi + Pi+1PiCi (4-23) = Gi+2 + Pi+2Gi+1 + Pi+2 Pi+1Gi + Pi+2 Pi+1PiCi = Gi+3 + Pi+3Gi+2 + Pi+3 Pi+2Gi+1 + Pi+3 Pi+2 Pi+1Gi + Pi+3 Pi+2 Pi+1PiCi Assuming that the maximum fan-in of the gates is not exceeded, each of these equations can be implemented in a two-level circuit so, if a change in Ci propagates to Cj (j = i + 1, i + 2, · · ·), it does so with a delay of two gates. Equations (4-23) are the carry-lookahead equations. If a circuit implements, for example, four of the equations, it is a 4-bit carry-lookahead circuit. Figure 4-8 shows a 4-bit parallel adder using a 4-bit carry-lookahead circuit. (The sum outputs are not shown.) After the generate and propagate outputs of the full adders are stable, if a change in C0 propagates to Ci (i = 1, 2, 3, or 4), it does so in two gate delays. Similarly, if a change in C1 propagates to Ci (i = 2, 3, or 4), it does so in two gate delays. In the 4-bit ripple-carry adder a change in C0 propagating to C4 requires 8 gate delays. FIGURE 4-8 4-Bit Adder with Carry-Lookahead A3 Full Adder © Cengage Learning 2014 G3 C4 A2 B3 C3 P3 Full Adder G2 A1 B2 C2 P2 Full Adder G1 A0 B1 C1 P1 4-bit Carry-Lookahead Circuit B0 Full Adder G0 P0 C0 The carry-lookahead circuit can be increased in size to reduce the delay in longer parallel adders; however, the gate fan-in increases linearly with the size of the carrylookahead circuit so the size is limited by the maximum fan-in available. For longer adders the carry-lookahead circuits can be cascaded. For example, a 16-bit parallel adder can be implemented using four 4-bit carry-lookahead circuits, as shown in Figure 4-9. Now the speed of the circuit is determined by the number of carrylookahead circuits required. In Figure 4-9 the propagation delay from C0 to C16 would be 8 gate delays; a 16-bit ripple-carry adder would have a delay of 32 gates. To reduce the delay of the adder without increasing the size of the carrylookahead circuit, a second level of carry-lookahead circuits can be connected to the first level carry-lookahead circuits. To illustrate this, the equations for the four carry-lookahead circuits in Figure 4-9 are written in same form as Equation (4-23). Applications of Boolean Algebra Minterm and Maxterm Expansions FIGURE 4-9 16-Bit Adder with Carry-Lookahead A12–15 B12–15 C0–3 Full Adders © Cengage Learning 2014 G12–15 C16 A8–11 B8–11 P12–15 C8–11 Full Adders G8–11 A4–7 P8–11 A0–3 B0–3 B4–7 C4–7 Full Adders G4–7 113 P4–7 C0–3 Full Adders G0–3 P0–3 C12 C8 C4 C0 4-bit 4-bit 4-bit 4-bit Carry-Lookahead Carry-Lookahead Carry-Lookahead Carry-Lookahead C4 = G0 + P0C0 where G0 = G3 + P3G2 + P3P2G1 + P3P2 P1G0 and P0 = P3P2 P1P0 C8 = G4 + P4C4 where G4 = G7 + P7G6 + P7P6G5 + P7P6P5G4 and P4 = P7P6P5P4 C12 = G8 + P8C8 where G8 = G11 + P11G10 + P11P10G9 + P11P10P9G8 and P8 = P11P10P9P8 C16 = G12 + P12C12 where G12 = G15 + P15G14 + P15P14G13 + P15P14P13G12 and P12 = P15P14P13P12 Now these equations for C4, C8, C12, and C16 can be written in terms of C0. C4 = C8 = C12 = C16 = G0 G4 G8 G12 + P0C0 + P4G0 + P4 P0C0 + P8G4 + P8 P4G0 + P8 P4 P0C0 + P12G8 + P12 P8G4 + P12 P8 P4G0 + P12 P8 P4 P0C0 These equations are the same as those for a 4-bit carry-lookahead circuit. The first level carry-lookahead circuits can be modified to produce Gi and Pi instead of Ci, i = 0, 4, 8, and 12. These provide inputs to a second level carry-lookahead circuit, as shown in Figure 4-10. Now the propagation delay from C0 to Ci, i = 4, 8, 12, and 16, is just two gate delays. A12–15 B12–15 FIGURE 4-10 16-Bit Adder with Second Level Carry-Lookahead C12–15 Full Adders © Cengage Learning 2014 G12–15 A8–11 B8–11 P12–15 C8–11 Full Adders G8–11 A4–7 P8–11 B4–7 C4–7 Full Adders G4–7 A0–3 P4–7 B0–3 C0–3 Full Adders G0–3 P0–3 C0 C8 C4 C12 4-bit 4-bit 4-bit Carry-Lookahead Carry-Lookahead Carry-Lookahead Carry-Lookahead G12 C16 P12 G8 P8 G4 P4 4-bit Carry-Lookahead Circuit G0 P0 114 Unit 4 Ci+1 = Gi + PiGi−1 + Pi Pi−1Gi−2 + Pi Pi−1Pi−2Gi−3 + Pi Pi−1Pi−2Pi−3Ci−3 C4 = G3 + P3G2 + P3P2G1 + P3P2P1G0 + P3P2P1P0C0 = G0 + P0C0 This expression can be expanded to express Ci+1 in terms of Ci−1. Ci+1 = Gi + Ci Pi + Gi = (Gi−1 + Ci−1Pi−1)Pi = Gi + PiGi−1 + Pi Pi−1Ci−1 This procedure can be continued to obtain Ci+1 Ci+1 Ci+1 Ci+1 = = = = Gi + Gi + Gi + Gi + CiPi PiGi−1 + Pi Pi−1Ci−1 PiGi−1 + Pi Pi−1Gi−2 + Pi Pi−1Pi−2Ci−2 PiGi−1 + Pi Pi−1Gi−2 + Pi Pi−1Pi−2Gi−3 + Pi Pi−1Pi−2Pi−3Ci−3 and so on. Problems 4.1 Represent each of the following sentences by a Boolean equation. (a) The company safe should be unlocked only when Mr. Jones is in the office or Mr. Evans is in the office, and only when the company is open for business, and only when the security guard is present. (b) You should wear your overshoes if you are outside in a heavy rain and you are wearing your new suede shoes, or if your mother tells you to. (c) You should laugh at a joke if it is funny, it is in good taste, and it is not offensive to others, or if it is told in class by your professor (regardless of whether it is funny and in good taste) and it is not offensive to others. (d) The elevator door should open if the elevator is stopped, it is level with the floor, and the timer has not expired, or if the elevator is stopped, it is level with the floor, and a button is pressed. 4.2 A flow rate sensing device used on a liquid transport pipeline functions as follows. The device provides a 5-bit output where all five bits are zero if the flow rate is less than 10 gallons per minute. The first bit is 1 if the flow rate is at least 10 gallons per minute; the first and second bits are 1 if the flow rate is at least 20 gallons per minute; the first, second, and third bits are 1 if the flow rate is at least 30 gallons per minute; and so on. The five bits, represented by the logical variables A, B, C, D, and E, are used as inputs to a device that provides two outputs Y and Z. (a) Write an equation for the output Y if we want Y to be 1 iff the flow rate is less than 30 gallons per minute. (b) Write an equation for the output Z if we want Z to be 1 iff the flow rate is at least 20 gallons per minute but less than 50 gallons per minute. Applications of Boolean Algebra Minterm and Maxterm Expansions 115 4.3 Given F1 = Σ m(0, 4, 5, 6) and F2 = Σ m(0, 3, 6, 7) find the minterm expression for F1 + F2. State a general rule for finding the expression for F1 + F2 given the minterm expansions for F1 and F2. Prove your answer by using the general form of the minterm expansion. 4.4 (a) How many switching functions of two variables (x and y) are there? (b) Give each function in truth table form and in reduced algebraic form. 4.5 A combinational circuit is divided into two subcircuits N1 and N2 as shown. The truth table for N1 is given. Assume that the input combinations ABC = 110 and ABC = 010 never occur. Change as many of the values of D, E, and F to don't-cares as you can without changing the value of the output Z. N1 A A 0 0 0 0 1 1 1 1 N2 D E B C F Z B 0 0 1 1 0 0 1 1 C 0 1 0 1 0 1 0 1 D 1 0 0 1 1 1 0 0 E 1 0 1 1 0 0 1 0 F 0 1 1 1 0 1 0 0 4.6 Work (a) and (b) with the following truth table: A 0 0 0 0 1 1 1 1 B 0 0 1 1 0 0 1 1 C 0 1 0 1 0 1 0 1 F 1 X 0 0 0 X 1 1 G 0 1 X 1 0 1 X 1 (a) Find the simplest expression for F, and specify the values of the don't-cares that lead to this expression. (b) Repeat (a) for G. (Hint: Can you make G the same as one of the inputs by properly choosing the values for the don't-care?) 4.7 Each of three coins has two sides, heads and tails. Represent the heads or tails status of each coin by a logical variable (A for the first coin, B for the second coin, and C for the third) where the logical variable is 1 for heads and 0 for tails. Write a logic function F(A, B, C) which is 1 iff exactly one of the coins is heads after a toss of the coins. Express F (a) as a minterm expansion. (b) as a maxterm expansion. 116 Unit 4 4.8 A switching circuit has four inputs as shown. A and B represent the first and second bits of a binary number N1. C and D represent the first and second bits of a binary number N2. The output is to be 1 only if the product N1 × N2 is less than or equal to 2. (a) Find the minterm expansion for F. (b) Find the maxterm expansion for F. Express your answers in both decimal notation and algebraic form. N1 N2 A B C D F 4.9 Given: F(a, b, c) = abc′ + b′. (a) Express F as a minterm expansion. (Use m-notation.) (b) Express F as a maxterm expansion. (Use M-notation.) (c) Express F′ as a minterm expansion. (Use m-notation.) (d) Express F′ as a maxterm expansion. (Use M-notation.) 4.10 Work Problem 4.9 using: F(a, b, c, d) = (a + b + d)(a′ + c)(a′ + b′ + c′)(a + b + c′ + d′) 4.11 (a) Implement a full subtracter using a minimum number of gates. (b) Compare the logic equations for the full adder and full subtracter. What is the relation between si and di? Between ci+1 and bi+1? 4.12 Design a circuit which will perform the following function on three 4-bit numbers: (X3 X2 X1 X0 + Y3Y2Y1Y0) − Z3Z2Z1Z0 It will give a result S3S2S1S0, a carry, and a borrow. Use eight full adders and any other type of gates. Assume that negative numbers are represented in 2's complement. 4.13 A combinational logic circuit has four inputs (A, B, C, and D) and one output Z. The output is 1 iff the input has three consecutive 0's or three consecutive 1's. For example, if A = 1, B = 0, C = 0, and D = 0, then Z = 1, but if A = 0, B = 1, C = 0, and D = 0, then Z = 0. Design the circuit using one four-input OR gate and four three-input AND gates. 4.14 Design a combinational logic circuit which has one output Z and a 4-bit input ABCD representing a binary number. Z should be 1 iff the input is at least 5, but is no greater than 11. Use one OR gate (three inputs) and three AND gates (with no more than three inputs each). 4.15 A logic circuit realizing the function f has four inputs A, B, C, and D. The three inputs A, B, and C are the binary representation of the digits 0 through 7 with A being the most-significant bit. The input D is an odd-parity bit, i.e., the value of D is such that Applications of Boolean Algebra Minterm and Maxterm Expansions 117 A, B, C, and D always contain an odd number of 1's. (For example, the digit 1 is represented by ABC = 001 and D = 0, and the digit 3 is represented by ABCD = 0111.) The function f has value 1 if the input digit is a prime number. (A number is prime if it is divisible only by itself and 1; 1 is considered to be prime and 0 is not.) (a) List the minterms and don't-care minterms of f in algebraic form. (b) List the maxterms and don't-care maxterms of f in algebraic form. 4.16 A priority encoder circuit has four inputs, x3, x2, x1, and x0. The circuit has three outputs: z, y1, and y0. If one of the inputs is 1, z is 1 and y1 and y0 represent a 2-bit, binary number whose value equals the index of the highest numbered input that is 1.

For example, if x2 is 1 and x3 is 0, then the outputs are z = 1 and y1 = 1 and y0 = 0. If all inputs are 0, z = 0 and y1 and y0 are don't-cares. (a) List in decimal form the minterms and don't-care minterms of each output. (b) List in decimal form the maxterms and don't-care maxterms of each output. 4.17 The 9's complement of a decimal digit d (0 to 9) is defined to be 9 − d. A logic circuit produces the 9's complement of an input digit where the input and output digits are represented in BCD. Label the inputs A, B, C, and D, and label the outputs W, X, Y and Z. (a) Determine the minterms and don't-care minterms for each of the outputs. (b) Determine the maxterms and don't-care maxterms for each of the outputs. 4.18 Repeat Problem 4.17 for the case where the input and output digits are represented using the 4-2-2-1 weighted code. (If only one weight of 2 is required for decimal digits less than 5, select the rightmost 2. In addition, select the codes so that W = A′, X = B′, Y = C′, and Z = D′. (There are two possible codes with these restrictions.) 4.19 Each of the following sentences has two possible interpretations depending on whether the AND or OR is done first. Write an equation for each interpretation. (a) The buzzer will sound if the key is in the ignition switch, and the car door is open, or the seat belts are not fastened. (b) You will gain weight if you eat too much, or you do not exercise enough, and your metabolism rate is too low. (c) The speaker will be damaged if the volume is set too high, and loud music is played, or the stereo is too powerful. (d) The roads will be very slippery if it snows, or it rains, and there is oil on the road. 4.20 A bank vault has three locks with a different key for each lock. Each key is owned by a different person. To open the door, at least two people must insert their keys into the assigned locks. The signal lines A, B, and C are 1 if there is a key inserted into lock 1, 2, or 3, respectively. Write an equation for the variable Z which is 1 iff the door should open. 4.21 A paper tape reader used as an input device to a computer has five rows of holes as shown. A hole punched in the tape indicates a logic 1, and no hole indicates a logic 0. As each hole pattern passes under the photocells, the pattern is translated into logic signals on lines A, B, C, D, and E. All patterns of holes indicate a valid character with two exceptions. A pattern consisting of none of the possible holes punched is not 118 Unit 4 used because it is impossible to distinguish between this pattern and the unpunched space between patterns. An incorrect pattern punched on the tape is erased by punching all five holes in that position. Therefore, a valid character punched on the tape will have at least one hole but will not have all five holes punched. (a) Write an equation for a variable Z which is 1 iff a valid character is being read. (b) Write an equation for a variable Y which is 1 iff the hole pattern being read has holes punched only in rows C and E. Photocells Variables A B C D E 4.22 A computer interface to a line printer has seven data lines that control the movement of the paper and the print head and determine which character to print. The data lines are labeled A, B, C, D, E, F, and G, and each represents a binary 0 or 1. When the data lines are interpreted as a 7-bit binary number with line A being the most significant bit, the data lines can represent the numbers 0 to 12710. The number 1310 is the command to return the print head to the beginning of a line, the number 1010 means to advance the paper by one line, and the numbers 3210 to 12710 represent printing characters. (a) Write an equation for the variable X which is 1 iff the data lines indicate a command to return the print head to the beginning of the line. (b) Write an equation for the variable Y which is 1 iff there is an advance paper command on the data lines. (c) Write an equation for the variable Z which is 1 iff the data lines indicate a printable character. (Hint: Consider the binary representations of the numbers 0–31 and 32–127 and write the equation for Z with only two terms.) 4.23 Given F1 = Π M(0, 4, 5, 6) and F2 = Π M(0, 4, 7), find the maxterm expansion for F1F2. State a general rule for finding the maxterm expansion of F1F2 given the maxterm expansions of F1 and F2. Prove your answer by using the general form of the maxterm expansion. 4.24 Given F1 = Π M(0, 4, 5, 6) and F2 = Π M(0, 4, 7), find the maxterm expansion for F1 + F2. State a general rule for finding the maxterm expansion of F1 + F2, given the maxterm expansions of F1 and F2. Prove your answer by using the general form of the maxterm expansion. 4.25 Four chairs are placed in a row: A B C D Applications of Boolean Algebra Minterm and Maxterm Expansions 119 Each chair may be occupied (1) or empty (0). Give the minterm and maxterm expansion for each logic function described. (a) F(A, B, C, D) is 1 iff there are no adjacent empty chairs. (b) G(A, B, C, D) is 1 iff the chairs on the ends are both empty. (c) H(A, B, C, D) is 1 iff at least three chairs are full. (d) J(A, B, C, D) is 1 iff there are more people sitting in the left two chairs than in the right two chairs. 4.26 Four chairs (A, B, C, and D) are placed in a circle: A next to B, B next to C, C next to D, and D next to A. Each chair may be occupied (1) or empty (0). Give the minterm and maxterm expansion for each of the following logic functions: (a) F(A, B, C, D) is 1 iff there are no adjacent empty chairs. (b) G(A, B, C, D) is 1 iff there are at least three adjacent empty chairs. (c) H(A, B, C, D) is 1 iff at least three chairs are full. (d) J(A, B, C, D) is 1 iff there are more people sitting in chairs A and B than chairs C and D. 4.27 Given f(a, b, c) = a(b + c′). (a) Express f as a minterm expansion (use m-notation). (b) Express f as maxterm expansion (use M-notation). (c) Express f ′ as a minterm expansion (use m-notation). (d) Express f ′ as a maxterm expansion (use M-notation). 4.28 Work Problem 4.27 using f(a, b, c, d) = acd + bd′ + a′c′d + ab′cd + a′b′cd′. 4.29 Find both the minterm expansion and maxterm expansion for the following functions, using algebraic manipulations: (a) f(A, B, C, D) = AB + A′CD (b) f(A, B, C, D) = (A + B + D′)(A′ + C)(C + D) 4.30 Given F′(A, B, C, D) = Σ m(1, 2, 6, 7, 13, 15). (a) Find the minterm expansion for F (both decimal and algebraic form). (b) Find the maxterm expansion for F (both decimal and algebraic form). 4.31 Repeat Problem 4.30 for F′(A, B, C, D) = Σ m(1, 2, 5, 6, 10, 15). 4.32 Work parts (a) through (d) with the given truth table. A 0 0 0 0 1 1 1 1 B 0 0 1 1 0 0 1 1 C 0 1 0 1 0 1 0 1 F1 1 X 0 0 0 X 0 1 F2 1 0 1 0 1 0 X X F3 0 0 X 1 1 1 X 1 F4 1 0 0 1 1 0 X X 120 Unit 4 (a) Find the simplest expression for F1, and specify the values for the don't-cares that lead to this expression. (b) Repeat for F2. (c) Repeat for F3. (d) Repeat for F4. 4.33 Work Problem 4.5 using the following circuits and truth table. Assume that the input combinations of ABC = 011 and ABC = 110 will never occur. N1 N2 D A B C Z E F A 0 0 0 0 1 1 1 1 B 0 0 1 1 0 0 1 1 C 0 1 0 1 0 1 0 1 D 1 0 0 0 0 0 0 1 E 1 1 0 0 1 0 0 0 F 0 0 1 0 0 1 1 1 4.34 Work Problem 4.7 for the following logic functions: (a) G1(A, B, C) is 1 iff all the coins landed on the same side (heads or tails). (b) G2(A, B, C) is 1 iff the second coin landed on the same side as the first coin. 4.35 A combinational circuit has four inputs (A, B, C, D) and three outputs (X, Y, Z). XYZ represents a binary number whose value equals the number of 1's at the input. For example if ABCD = 1011, XYZ = 011. (a) Find the minterm expansions for X, Y, and Z. (b) Find the maxterm expansions for Y and Z. 4.36 A combinational circuit has four inputs (A, B, C, D) and four outputs (W, X, Y, Z). WXYZ represents an excess-3 coded number whose value equals the number of 1's at the input. For example, if ABCD = 1101, WXYZ = 0110. (a) Find the minterm expansions for X, Y, and Z. (b) Find the maxterm expansions for Y and Z. 4.37 A combinational circuit has four inputs (A, B, C, D), which represent a binarycoded-decimal digit. The circuit has two groups of four outputs—S, T, U, V, and W, X, Y, Z. Each group represents a BCD digit. The output digits represent a decimal number which is five times the input number. For example, if ABCD = 0111, the outputs are 0011 0101. Assume that invalid BCD digits do not occur as inputs. (a) Construct the truth table. (b) Write down the minimum expressions for the outputs by inspection of the truth table. (Hint: Try to match output columns in the table with input columns.) 4.38 Work Problem 4.37 where the BCD outputs represent a decimal number that is 1 more than four times the input number. For example, if ABCD = 0011, the outputs are 0001 0011. Applications of Boolean Algebra Minterm and Maxterm Expansions 121 4.39 Design a circuit which will add a 4-bit binary number to a 5-bit binary number. Use five full adders. Assume negative numbers are represented in 2's complement. (Hint: How do you make a 4-bit binary number into a 5-bit binary number, without making a negative number positive or a positive number negative? Try writing down the representation for –3 as a 3-bit 2's complement number, a 4-bit 2's complement number, and a 5-bit 2's complement number. Recall that one way to find the 2's complement of a binary number is to complement all bits to the left of the first 1.) 4.40 A half adder is a circuit that adds two bits to give a sum and a carry. Give the truth table for a half adder, and design the circuit using only two gates. Then design a circuit which will find the 2's complement of a 4-bit binary number. Use four half adders and any additional gates. (Hint: Recall that one way to find the 2's complement of a binary number is to complement all bits, and then add 1.) 4.41 (a) Write the switching function f(x, y) = x + y as a sum of minterms and as a product of maxterms. (b) Consider the Boolean algebra of four elements 5 0, 1, a, b 6 specified by the following operation tables and the Boolean function f(x, y) = ax + by where a and b are two of the elements in the Boolean algebra. Write f(x, y) in a sum-ofminterms form. (c) Write the Boolean function of part (b) in a product-of-maxterms form. (d) Give a table of combinations for the Boolean function of part (b). (Note: The table of combinations has 16 rows, not just 4.) (e) Which four rows of the table of combinations completely specify the function of part (b)? Verify your answer. 0 1 a b ′ 1 0 b a + 0 1 a b 0 0 1 a b 1 1 1 1 1 a a 1 a 1 b b 1 1 b · 0 1 a b 0 0 0 0 0 1 0 1 a b a 0 a a 0 b 0 b 0 b 4.42 (a) If m1 and m2 are minterms of n variables, prove that m1 + m2 = m1 ⊕ m2. (b) Prove that any switching function can be written as the exclusive OR sum of products where each product does not contain a complemented literal. (Hint: Start with the function written as a sum of minterms and use part (a).) 4.43 (a) Show that the full adder of Figure 4-5 can be implemented using two 2-input exclusive OR gates and three 2-input NAND gates. (Hint: Rewrite Equation (4-2) in terms of X ⊕ Y.) (b) Compare the maximum addition time of the ripple-carry adder of Figure 4-3 using the full adder of part (a) versus the full adder of Figure 4-5 assuming the same gate types are used in both. 4.44 Show that a full subtractor can be implemented using two 2-input exclusive OR gates, one inverter, and three 2-input NOR gates. (Hint: Write the borrow out equation in product-of-sums form.) 122 Unit 4 4.45 The full adder of Figure 4-5 is modified by adding two control inputs, E1 and E0, and implemented as shown in the figure below. (a) For each combination of values for E1 and E0, give the algebraic expression for the outputs of the full adder. (b) Assume this modified full adder is used in the parallel adder of Figure 4-3. For each combination of values for E1 and E0, specify the function (Add, Exclusive OR, etc.) performed by the parallel adder. E1 E0 ai bi ai G1 si bi E1′ ci G2 E0 ai bi ci ci+1 bi ci 4.46 Redo Problem 4.45 if gates G1 and G2 are NAND gates rather than AND gates. 4.47 Redo Problem 4.45 if gates G1 and G2 are NOR gates rather than AND gates and an inverter is inserted in the ci input of G2. 4.48 Redo Problem 4.45 if gates G1 and G2 are OR gates rather than AND gates and an inverter is inserted in the ci input of G2. UNIT Karnaugh Maps 5 Objectives 1. Given a function (completely or incompletely specified) of three to five variables, plot it on a Karnaugh map. The function may be given in minterm, maxterm, or algebraic form. 2. Determine the essential prime implicants of a function from a map. 3. Obtain the minimum sum-of-products or minimum product-of-sums form of a function from the map. 4. Determine all of the prime implicants of a function from a map. 5. Understand the relation between operations performed using the map and the corresponding algebraic operations. 123 124 Unit 5 Study Guide In this unit we will study the Karnaugh (pronounced “car-no”) map. Just about any type of algebraic manipulation we have done so far can be facilitated by using the map, provided the number of variables is small. 1. Study Section 5.1, Minimum Forms of Switching Functions. (a) Define a minimum sum of products. (b) Define a minimum product of sums. 2. Study Section 5.2, Two- and Three-Variable Karnaugh Maps. (a) Plot the given truth table on the map. Then, loop two pairs of 1's on the map and write the simplified form of F. PQ 00 01 10 11 F 1 1 0 1 P 0 Q 1 0 1 F= F Now simplify F algebraically and verify that your answer is correct. (b) F(a, b, c) is plotted below. Find the truth table for F. a 0 1 00 0 1 01 1 1 11 0 1 10 1 0 bc F abc 000 001 010 011 100 101 110 111 F Karnaugh Maps 125 (c) Plot the following functions on the given Karnaugh maps: F1(R, S, T) = Σ m(0, 1, 5, 6) F2(R, S, T) = Π M(2, 3, 4, 7) 0 1 0 00 00 01 01 11 11 10 10 1 Why are the two maps the same? (d) Plot the following function on the given map: f(x, y, z) = z′ + x′z + yz Do not make a minterm expansion or a truth table before plotting. x yz 0 1 00 01 11 10 (e) For a three-variable map, which squares are “adjacent” to square 2? ____________ (f) What theorem is used when two terms in adjacent squares are combined? (g) What law of Boolean algebra justifies using a given 1 on a map in two or more loops? 126 Unit 5 (h) Each of the following solutions is not minimum. a a 0 bc 1 00 1 01 1 11 1 0 bc f = ab′ + abc 10 1 00 1 01 1 11 1 1 10 1 1 g = a′ + ab In each case, change the looping on the map so that the minimum solution is obtained. (i) Work Problem 5.3. (j) Find two different minimum sum-of-products expressions for the function G, which is plotted below. a bc 00 a 0 1 1 1 00 1 01 1 11 1 10 1 01 11 1 10 1 bc 0 1 1 1 1 G= 1 G= G Study Section 5.3, Four-Variable Karnaugh Maps. (a) Note the locations of the minterms on three- and four-variable maps (Figures 5-3(b) and 5-10). Memorize this ordering. This will save you a lot of time when you are plotting Karnaugh maps. This ordering is valid only for the order of the variables given. If we label the maps as shown below, fill in the locations of the minterms: BC 00 A 01 11 10 CD 00 AB 0 00 1 01 11 10 01 11 10 Karnaugh Maps 127 (b) Given the following map, write the minterm and maxterm expansions for F in decimal form: ab 00 cd 01 00 01 1 11 10 1 1 1 1 1 1 11 F= 1 1 10 F= (c) Plot the following functions on the given maps: (1) f (w, x, y, z) = Σ m(0, 1, 2, 5, 7, 8, 9, 10, 13, 14) (2) f (w, x, y, z) = x′z′ + y′z + w′xz + wyz′ wx 00 yz 01 11 10 wx 00 yz 00 00 01 01 11 11 10 10 01 11 10 Your answers to (1) and (2) should be the same. (d) For a four-variable map, which squares are adjacent to square 14? ________ To square 8? _________ (e) When we combine two adjacent 1's on a map, this corresponds to applying the theorem xy′ + xy = x to eliminate the variable in which the two terms differ. Thus, looping the two 1's as indicated on the following map is equivalent to combining the corresponding minterms algebraically: ab cd 00 01 10 1 00 01 1 11 1 10 11 1 a′b′c′d + a′b′cd = b′c′d [The term b′c′d can be read directly from the map because it spans the first and last columns (b′) and because it is in the second row (c′d).] 1 1 128 Unit 5 Loop two other pairs of adjacent 1's on this map and state the algebraic equivalent of looping these terms. Now read the loops directly off the map and check your algebra. (f) When we combine four adjacent 1's on a map (either four in a line or four in a square) this is equivalent to applying xy + xy′ = x three times: ab 00 cd 01 00 1 01 1 11 10 11 1 1 1 10 1 1 1 a′b′cd + a′bcd′ + ab′cd + ab′cd′ = a′b′c + ab′c = b′c Loop the other four 1's on the map and state the algebraic equivalent. (g) For each of the following maps, loop a minimum number of terms which will cover all of the 1's. ab ab 01 11 00 1 1 01 1 1 cd 11 10 00 1 10 cd 00 11 1 10 1 00 1 01 1 11 1 1 1 10 1 f1 01 1 1 1 1 f2 (For each part you should have looped two groups of four 1's and two groups of two 1's). Write down the minimum sum-of-products expression for f1 and f2 from these maps. f1 = ______________________________ f2 = ______________________________ (h) Why is it not possible to combine three or six minterms together rather than just two, four, eight, etc.? Karnaugh Maps 129 (i) Note the procedure for deriving the minimum product of sums from the map. You will probably make fewer mistakes if you write down f ′ as a sum of products first and then complement it, as illustrated by the example in Figure 5-14. (j) Work Problems 5.4 and 5.5. 4. Study Section 5.4, Determination of Minimum Expressions Using Essential Prime Implicants. (a) For the map of Figure 5-15, list three implicants of F other than those which are labeled. For the same map, is ac′d′ a prime implicant of F? Why or why not? (b) For the given map, are any of the circled terms prime implicants? Why or why not? 5. AB CD 00 01 11 10 1 00 01 1 1 11 1 1 10 1 1 1 Study Figure 5-18 carefully and then answer the following questions for the given map: (a) How many 1's are adjacent to m0? (b) Are all these 1's covered by a single prime implicant? AB 00 CD 00 01 1 1 0 (c) From your answer to (b), can you determine whether B′C′ is essential? 01 (d) How many 1's are adjacent to m9? 11 (e) Are all of these 1's covered by a single prime implicant? 10 11 10 1 4 1 8 1 1 1 9 1 3 1 7 1 2 1 6 10 (f) From your answer to (e), is B′C′ essential? (g) How many 1's are adjacent to m7? (h) Why is A′C essential? (i) Find two other essential prime implicants and tell which minterm makes them essential. 130 Unit 5 6. (a) How do you determine if a prime implicant is essential using a Karnaugh map? (b) For the following map, why is A′B′not essential? Why is BD′ essential? Is A′D′ essential? Why? Is BC′ essential? Why? Is B′CD essential? Why? Find the minimum sum of products. (c) Work Programmed Exercise 5.1. (d) List all 1's and X's that are adjacent to 10. AB 00 CD 01 11 10 00 10 14 112 8 01 X1 15 X13 9 11 3 X7 115 111 10 2 6 X14 10 Why is A′C′ an essential prime implicant? List all 1's and X's adjacent to 115. AB 00 CD 01 11 00 1 1 1 01 1 1 1 11 1 10 1 10 1 1 1 Karnaugh Maps 131 Based on this list, why can you not find an essential prime implicant that covers 115? Does this mean that there is no essential prime implicant that covers 115? What essential prime implicant covers 111? Can you find an essential prime implicant that covers 112? Explain. Find two prime implicants that cover 112. Give two minimum expressions for F. (e) Work Problem 5.6. (f) If you have a copy of the LogicAid program available, use the Karnaugh map tutorial mode to help you learn to find minimum solutions from Karnaugh maps. This program will check your work at each step to make sure that you loop the terms in the correct order. It also will check your final answer. Work Problem 5.7 using the Karnaugh map tutor. 7. (a) In Example 4, page 107, we derived the following function: Z = Σ m(0, 3, 6, 9) + Σ d(10, 11, 12, 13, 14, 15) Plot Z on the given map using X's to represent don't-care terms. AB 00 CD 01 11 10 00 01 11 10 Z (b) Show that the minimum sum of products is Z = A′B′C′D′ + B′CD + AD + BCD′ Which four don't-care minterms were assigned the value 1 when forming your solution? 132 Unit 5 (c) Show that the minimum product of sums for Z is Z = (B′ + C)(B′ + D′)(A′ + D)(A + C + D′)(B + C′ + D) Which one don't-care term of Z was assigned the value 1 when forming your solution? (d) Work Problem 5.8. 8. Study Section 5.5, Five-Variable Karnaugh Maps. (a) The figure below shows a three-dimensional five-variable map. Plot the 1's and loops on the corresponding two-dimensional map, and give the minimum sum-of-products expression for the function. BC 00 DE 00 A= 1 01 1 1 01 11 BC DE 10 1 1 00 01 11 10 00 11 10 01 1 A 1 0 1 11 A= 0 1 1 10 F= (b) On a five-variable map (Figure 5-21), what are the five minterms adjacent to minterm 24? (c) Work through all of the examples in this section carefully and make sure that you understand all of the steps. (d) Two minimum solutions are given for Figure 5-24. There is a third minimum sum-of-products solution. What is it? (e) Work Programmed Exercise 5.2. Karnaugh Maps (f) BC DE 00 01 16 X 00 11 20 4 21 12 29 8 25 1 X 01 1 1 19 5 23 13 31 1 1 3 18 1 7 22 1 X 15 30 1 2 9 27 1 X 11 10 24 1 0 A 1 0 10 28 X 17 133 X 6 11 26 X 14 10 Find the three 1's and X's adjacent to 118. Can these all be looped with a single loop? Find the 1's and X's adjacent to 124. Loop the essential prime implicant that covers 124. Find the 1's and X's adjacent to 13. Loop the essential prime implicant that covers 13. Can you find an essential prime implicant that covers 122? Explain. Find and loop two more essential prime implicants. Find three ways to cover the remaining 1 on the map and give the corresponding minimum solutions. (g) If you have the LogicAid program available, work Problem 5.9, using the Karnaugh map tutor. 9. Study Section 5.6, Other Uses of Karnaugh Maps. Refer to Figure 5-8 and note that a consensus term exists if there are two adjacent, but nonoverlapping prime implicants. Observe how this principle is applied in Figure 5-26. 10. Work Problems 5.10, 5.11, 5.12, and 5.13. When deriving the minimum solution from the map, always write down the essential prime implicants first. If you do not, it is quite likely that you will not get the minimum solution. In addition, make sure you can find all of the prime implicants from the map (see Problem 5.10(b)). 11. Review the objectives. Karnaugh Maps Switching functions can generally be simplified by using the algebraic techniques described in Unit 3. However, two problems arise when algebraic procedures are used: 1. 2. The procedures are difficult to apply in a systematic way. It is difficult to tell when you have arrived at a minimum solution. The Karnaugh map method studied in this unit and the Quine-McCluskey procedure studied in Unit 6 overcome these difficulties by providing systematic methods for simplifying switching functions. The Karnaugh map is an especially useful tool for simplifying and manipulating switching functions of three or four variables, but it can be extended to functions of five or more variables. Generally, you will find the Karnaugh map method is faster and easier to apply than other simplification methods. 5.1 Minimum Forms of Switching Functions When a function is realized using AND and OR gates, the cost of realizing the function is directly related to the number of gates and gate inputs used. The Karnaugh map techniques developed in this unit lead directly to minimum cost two-level circuits composed of AND and OR gates. An expression consisting of a sum of product terms corresponds directly to a two-level circuit composed of a group of AND gates feeding a single OR gate (see Figure 2-5). Similarly, a product-of-sums expression corresponds to a two-level circuit composed of OR gates feeding a single AND gate (see Figure 2-6). Therefore, to find minimum cost two-level AND-OR gate circuits, we must find minimum expressions in sum-of-products or product-ofsums form. A minimum sum-of-products expression for a function is defined as a sum of product terms which (a) has a minimum number of terms and (b) of all those expressions which have the same minimum number of terms, has a minimum number of literals. The minimum sum of products corresponds directly to a minimum two-level gate circuit which has (a) a minimum number of gates and (b) a minimum number 134 Karnaugh Maps 135 of gate inputs. Unlike the minterm expansion for a function, the minimum sum of products is not necessarily unique; that is, a given function may have two different minimum sum-of-products forms, each with the same number of terms and the same number of literals. Given a minterm expansion, the minimum sum-of-products form can often be obtained by the following procedure: 1. 2. Combine terms by using the uniting theorem XY′ + XY = X. Do this repeatedly to eliminate as many literals as possible. A given term may be used more than once because X + X = X. Eliminate redundant terms by using the consensus theorem or other theorems. Unfortunately, the result of this procedure may depend on the order in which terms are combined or eliminated so that the final expression obtained is not necessarily minimum. Example Find a minimum sum-of-products expression for F Σ m (0, 1, 2, 5, 6, 7) F(a, b, c) a′b′c a′b′c′ a′b′ a′bc′ b′c ab′c bc′ abc′ abc ab (5-1) None of the terms in the above expression can be eliminated by consensus. However, combining terms in a different way leads directly to a minimum sum of products: F a′b′c a′b′c′ a′b′ a′bc′ ab′c bc′ abc′ ac abc (5-2) If the uniting theorem is applied to all possible pairs of minterms, six two-literal products are obtained: a′b′, a′c′, b′c, bc′, ac, ab. Then, the consensus theorem can be applied to obtain a second minimal solution: a′c′ + b′c + ab (5-3) A minimum product-of-sums expression for a function is defined as a product of sum terms which (a) has a minimum number of terms, and (b) of all those expressions which have the same number of terms, has a minimum number of literals. Unlike the maxterm expansion, the minimum product-of-sums form of a function is not necessarily unique. Given a maxterm expansion, the minimum product of sums can often be obtained by a procedure similar to that used in the minimum sumof-products case, except that the uniting theorem (X + Y)(X + Y′) = X is used to combine terms. 136 Unit 5 Example (A B′ C D′)(A B′ C′ (A B′ D′) (A B′ D′) (A B′ D′)(A D′)(C′ B′ (A (A D) C′ B′ B′ D)(A′ C′) C′) B′ (B′ C′ D)(A C′ D) (C′ D) B (B C′ C′ D)(A′ B C′ D) D) eliminate by consensus (5-4) The uniting theorem XY′ + XY = X can be applied to minterms and products where the minterms and products are represented in algebraic notation or binary notation. The first four-variable example below illustrates this for minterms and the second for products containing three literals. The dash indicates a missing variable. ab′cd′ + ab′cd = ab′c 1 0 1 0 + 1 0 1 1 = 101– ab′c + abc = ac 1 0 1– + 111– = 1–1– Note that minterms only combine if they differ in one variable, and products only combine if they have dashes in the same position (same missing variables) and differ in one other variable. The examples below do not combine. ab′cd′ + ab′c′d (will not combine) 10 10 + 1001 ab′c + abd (will not combine) 101– + 11–1 The Karnaugh maps introduced next arrange the minterms of a function so that it is easy to recognize visually when the simplification theorem applies to two minterms, two products with one missing variable, two products with two missing variables, etc. 5.2 Two- and Three-Variable Karnaugh Maps Just like a truth table, the Karnaugh map of a function specifies the value of the function for every combination of values of the independent variables. A two-variable Karnaugh map is shown. The values of one variable are listed across the top of the map, and the values of the other variable are listed on the left side. Each square of the map corresponds to a pair of values for A and B as indicated. Karnaugh Maps 137 A 0 B A = 0, B = 0 A = 0, B = 1 1 0 A = 1, B = 0 1 A = 1, B = 1 Figure 5-1 shows the truth table for a function F and the corresponding Karnaugh map. Note that the value of F for A = B = 0 is plotted in the upper left square, and the other map entries are plotted in a similar way in Figure 5-1(b). Each 1 on the map corresponds to a minterm of F. We can read the minterms from the map just like we can read them from the truth table. A 1 in square 00 of Figure 5-1(c) indicates that A′B′ is a minterm of F. Similarly, a 1 in square 01 indicates that A′B is a minterm. Minterms in adjacent squares of the map can be combined since they differ in only one variable. Thus, A′B′ and A′B combine to form A′, and this is indicated by looping the corresponding 1's on the map in Figure 5-1(d). FIGURE 5-1 © Cengage Learning 2014 A B AB F 0 1 0 1 1 1 0 0 0 0 1 1 0 A 0 1 1 0 B 0 A 0 1 1 0 1 0 1 0 1 0 1 1 0 A′B′ + A′B = A′ A′B′ 1 0 B 1 1 0 A′B F = A′B′ + A′B (b) (a) F = A′ (c) (d) Figure 5-2 shows a three-variable truth table and the corresponding Karnaugh map (see Figure 5-27 for an alternative way of labeling maps). The value of one variable (A) is listed across the top of the map, and the values of the other two variables (B, C) are listed along the side of the map. The rows are labeled in the sequence 00, 01, 11, 10 so that values in adjacent rows differ in only one variable. For each combination of values of the variables, the value of F is read from the truth table and plotted in the appropriate map square. For example, for the input combination ABC = 001, the value F = 0 is plotted in the square for which A = 0 and BC = 01. For the combination ABC = 110, F = 1 is plotted in the A = 1, BC = 10 square. FIGURE 5-2 Truth Table and Karnaugh Map for Three-Variable Function © Cengage Learning 2014 ABC F 0 0 0 0 1 1 1 1 0 0 1 1 1 0 1 0 0 0 1 1 0 0 1 1 0 1 0 1 0 1 0 1 (a) A BC 0 1 00 1 01 0 0 11 1 0 10 1 1 ABC = 001, F = 0 F (b) ABC = 110, F = 1 138 Unit 5 Figure 5-3 shows the location of the minterms on a three-variable map. Minterms in adjacent squares of the map differ in only one variable and therefore can be combined using the uniting theorem XY′ + XY = X. For example, minterm 011 (a′bc) is adjacent to the three minterms with which it can be combined—001 (a′b′c), 010 (a′bc′), and 111 (abc). In addition to squares which are physically adjacent, the top and bottom rows of the map are defined to be adjacent because the corresponding minterms in these rows differ in only one variable. Thus 000 and 010 are adjacent, and so are 100 and 110. FIGURE 5-3 Location of Minterms on a Three-Variable Karnaugh Map a a 0 1 00 000 100 01 001 101 11 011 111 10 010 110 bc 0 1 00 0 4 01 1 5 11 3 7 10 2 6 bc © Cengage Learning 2014 100 is adjacent to 110 (a) Binary notation (b) Decimal notation Given the minterm expansion of a function, it can be plotted on a map by placing 1's in the squares which correspond to minterms of the function and 0's in the remaining squares (the 0's may be omitted if desired). Figure 5-4 shows the plot of F(a, b, c) = m1 + m3 + m5. If F is given as a maxterm expansion, the map is plotted by placing 0's in the squares which correspond to the maxterms and then by filling in the remaining squares with 1's. Thus, F(a, b, c) = M0M2M4M6M7 gives the same map as Figure 5-4. FIGURE 5-4 Karnaugh Map of F(a, b, c) = Σ m(1, 3, 5) = Π M(0, 2, 4, 6, 7) © Cengage Learning 2014 a bc 00 0 1 0 0 0 01 1 4 1 1 11 1 5 0 3 10 0 7 0 2 6 Figure 5-5 illustrates how product terms can be plotted on Karnaugh maps. To plot the term b, 1's are entered in the four squares of the map where b = 1. The term bc′ is 1 when b = 1 and c = 0, so 1's are entered in the two squares in the bc = 10 row. The term ac′ is 1 when a = 1 and c = 0, so 1's are entered in the a = 1 column in the rows where c = 0. Karnaugh Maps FIGURE 5-5 Karnaugh Maps for Product Terms a bc a a 0 1 bc 0 1 0 bc 00 00 00 01 01 01 11 1 1 139 a = 1 in this column 1 © Cengage Learning 2014 b = 1 in these rows 11 1 1 11 10 1 1 10 1 b 1 c = 0 in these rows 1 10 ac′ bc′ If a function is given in algebraic form, it is unnecessary to expand it to minterm form before plotting it on a map. If the algebraic expression is converted to sum-ofproducts form, then each product term can be plotted directly as a group of 1's on the map. For example, given that f (a, b, c) = abc′ + b′c + a′ we would plot the map as follows: a bc 1. The term abc′ is 1 when a = 1 and bc = 10, so we place a 1 in the square which corresponds to the a = 1 column and the bc = 10 row of the map. 2. The term b′c is 1 when bc = 01, so we place 1's in both squares of the bc = 01 row of the map. 3. The term a′ is 1 when a = 0, so we place 1's in all the squares of the a = 0 column of the map. (Note: Since there already is a 1 in the abc = 001 square, we do not have to place a second 1 there because x + x = x.) 0 00 1 01 1 11 10 1 1 1 1 abc′ Figure 5-6 illustrates how a simplified expression for a function can be derived using a Karnaugh map. The function to be simplified is first plotted on a Karnaugh map in Figure 5-6(a). Terms in adjacent squares on the map differ in only one variable and can be combined using the uniting theorem XY′ + XY = X. Thus a′b′c and a′bc combine to form a′c, and a′b′c and ab′c combine to form b′c, as shown in Figure 5-6(b). A loop around a group of minterms indicates that these terms have been combined. The looped terms can be read directly off the map. Thus, for Figure 5-6(b), term T1 is in the a = 0 (a′) column, and it spans the rows where c = 1, so T1 = a′c. Note that b has been eliminated because the two minterms in T1 differ in the variable b. Similarly, the term T2 is in the bc = 01 row so T2 = b′c, and a has been eliminated because T2 spans the a = 0 and a = 1 columns. Thus, the minimum sum-of-products form for F is a′c + b′c. 140 Unit 5 FIGURE 5-6 Simplification of a Three-Variable Function © Cengage Learning 2014 a a 0 bc 1 bc 00 0 1 1 1 00 01 1 1 1 1 T1 01 = a′b′c + a′bc = a′c 11 1 1 10 T2 = a′b′c + ab′c = b′c 1 10 F = Σ m(1, 3, 5) F = a′c + b′c (b) Simplified form of F (a) Plot of minterms The map for the complement of F (Figure 5-7) is formed by replacing 0's with 1's and 1's with 0's on the map of F. To simplify F′, note that the terms in the top row combine to form b′c′, and the terms in the bottom row combine to form bc′. Because b′c′ and bc′ differ in only one variable, the top and bottom rows can then be combined to form a group of four 1's, thus eliminating two variables and leaving T1 = c′. The remaining 1 combines, as shown, to form T2 = ab, so the minimum sum- of-products form for F′ is c′ + ab. FIGURE 5-7 Complement of Map in Figure 5-6(a) a 0 1 00 1 1 01 0 0 11 0 1 10 1 1 bc © Cengage Learning 2014 T1 = b′c′ + bc′ = c′ T2 = ab The Karnaugh map can also illustrate the basic theorems of Boolean algebra. Figure 5-8 illustrates the consensus theorem, XY + X′Z + YZ = XY + X′Z. Note that the consensus term (YZ) is redundant because its 1's are covered by the other two terms. FIGURE 5-8 Karnaugh Maps that Illustrate the Consensus Theorem x yz 0 x yz 1 00 © Cengage Learning 2014 1 00 01 1 11 1 yz (consensus term) x′z 10 0 1 1 xy xy + x′z + yz = xy + x′z 01 1 11 1 10 1 1 Karnaugh Maps 141 If a function has two or more minimum sum-of-products forms, all of these forms can be determined from a map. Figure 5-9 shows the two minimum solutions for F = Σ m(0, 1, 2, 5, 6, 7). FIGURE 5-9 Function with Two Minimum Forms a a 0 bc 00 1 01 1 1 0 bc 00 1 01 1 11 1 11 10 1 1 0 1 © Cengage Learning 2014 11 10 1 1 F = a′b′ + bc′ + ac 1 1 1 1 F = a′c′ + b′c + ab 5.3 Four-Variable Karnaugh Maps Figure 5-10 shows the location of minterms on a four-variable map. Each minterm is located adjacent to the four terms with which it can combine. For example, m5 (0101) could combine with m1 (0001), m4 (0100), m7 (0111), or m13 (1101) because it differs in only one variable from each of the other minterms. The definition of adjacent squares must be extended so that not only are top and bottom rows adjacent as in the three-variable map, but the first and last columns are also adjacent. This requires numbering the columns in the sequence 00, 01, 11, 10 so that minterms 0 and 8, 1 and 9, etc., are in adjacent squares. FIGURE 5-10 Location of Minterms on Four-Variable Karnaugh Map © Cengage Learning 2014 AB 00 CD 01 11 10 00 0 4 12 8 01 1 5 13 9 11 3 7 15 11 10 2 6 14 10 We will now plot the following four-variable expression on a Karnaugh map (Figure 5-11): f (a, b, c, d) = acd + a′b + d′ The first term is 1 when a = c = d = 1, so we place 1's in the two squares which are in the a = 1 column and cd = 11 row. The term a′b is 1 when ab = 01, so we place four 1's in the ab = 01 column. Finally, d′ is 1 when d = 0, so we place eight 1's in the two rows for which d = 0. (Duplicate 1's are not plotted because 1 + 1 = 1.) 142 Unit 5 FIGURE 5-11 Plot of acd + a′b + d′ ab cd 00 00 01 11 10 1 1 1 1 © Cengage Learning 2014 a′b 01 1 1 1 1 1 1 1 1 d′ 1 10 acd Next, we will simplify the functions f1 and f2 given in Figure 5-12. Because the functions are specified in minterm form, we can determine the locations of the 1's on the map by referring to Figure 5-10. After plotting the maps, we can then combine adjacent groups of 1's. Minterms can be combined in groups of two, four, or eight to eliminate one, two, or three variables, respectively. In Figure 5-12(a), the pair of 1's in the ab = 00 column and also in the d = 1 rows represents a′b′d. The group of four 1's in the b = 1 columns and c = 0 rows represents bc′. FIGURE 5-12 Simplification of Four-Variable Functions ab cd ab 00 00 © Cengage Learning 2014 01 1 01 11 1 1 1 1 1 10 cd 00 bc′ 00 01 11 1 10 1 01 a′bd c a′b′d 11 1 1 10 ab′cd′ Four corner terms combine to give b′d′ 1 11 1 1 1 1 10 1 1 1 1 f1 = Σ m(1, 3, 4, 5, 10, 12, 13) = bc′ + a′b′d + ab′cd′ f2 = Σ m(0, 2, 3, 5, 6, 7, 8, 10, 11, 14, 15) = c + b′d′ + a′bd (a) (b) In Figure 5-12(b), note that the four corner 1's span the b = 0 columns and d = 0 rows and, therefore, can be combined to form the term b′d′. The group of eight 1's covers both rows where c = 1 and, therefore, represents the term c. The pair of 1's which is looped on the map represents the term a′bd because it is in the ab = 01 column and spans the d = 1 rows. The Karnaugh map method is easily extended to functions with don't-care terms. The required minterms are indicated by 1's on the map, and the don't-care minterms are indicated by X's. When choosing terms to form the minimum sum of products, all Karnaugh Maps 143 the 1's must be covered, but the X's are only used if they will simplify the resulting expression. In Figure 5-13, the only don't-care term used in forming the simplified expression is 13. FIGURE 5-13 Simplification of an Incompletely Specified Function © Cengage Learning 2014 ab 00 cd 01 11 10 X 00 01 1 1 11 1 1 X 1 X 10 f = Σ m(1, 3, 5, 7, 9) + Σ d(6, 12, 13) = a′d + c′d The use of Karnaugh maps to find a minimum sum-of-products form for a function has been illustrated in Figures 5-1, 5-6, and 5-12. A minimum product of sums can also be obtained from the map. Because the 0's of f are 1's of f ′, the minimum sum of products for f ′ can be determined by looping the 0's on a map of f . The complement of the minimum sum of products for f ′ is then the minimum product of sums for f . The following example illustrates this procedure for f = x′z′ + wyz + w′y′z′ + x′y First, the 1's of f are plotted in Figure 5-14. Then, from the 0's, f ′ = y′z + wxz′ + w′xy and the minimum product of sums for f is f = (y + z′)(w′ + x′ + z)(w + x′ + y′) FIGURE 5-14 © Cengage Learning 2014 wx 00 yz 01 11 10 00 1 1 0 1 01 0 0 0 0 11 1 0 1 1 10 1 0 0 1 144 Unit 5 5.4 Determination of Minimum Expressions Using Essential Prime Implicants Any single 1 or any group of 1's which can be combined together on a map of the function F represents a product term which is called an implicant of F (see Section 6.1 for a formal definition of implicant and prime implicant). Several implicants of F are indicated in Figure 5-15. A product term implicant is called a prime implicant if it cannot be combined with another term to eliminate a variable. In Figure 5-15, a′b′c, a′cd′, and ac′ are prime implicants because they cannot be combined with other terms to eliminate a variable. On the other hand, a′b′c′d′ is not a prime implicant because it can be combined with a′b′cd′ or ab′c′d′. Neither abc′, nor ab′c′is a prime implicant because these terms can be combined together to form ac′. FIGURE 5-15 ab cd © Cengage Learning 2014 00 00 01 1 11 10 1 1 1 1 a′b′c′d′ 01 ac′ ab′c′ abc′ a′b′c 11 1 10 1 1 a′cd′ All of the prime implicants of a function can be obtained from a Karnaugh map. A single 1 on a map represents a prime implicant if it is not adjacent to any other 1's. Two adjacent 1's on a map form a prime implicant if they are not contained in a group of four 1's; four adjacent 1's form a prime implicant if they are not contained in a group of eight 1's, etc. The minimum sum-of-products expression for a function consists of some (but not necessarily all) of the prime implicants of a function. In other words, a sum-ofproducts expression containing a term which is not a prime implicant cannot be minimum. This is true because if a nonprime term were present, the expression could be simplified by combining the nonprime term with additional minterms. In order to find the minimum sum of products from a map, we must find a minimum number of prime implicants which cover all of the 1's on the map. The function plotted in Figure 5-16 has six prime implicants. Three of these prime implicants cover all of the 1's on the map, and the minimum solution is the sum of these three prime implicants. The shaded loops represent prime implicants which are not part of the minimum solution. When writing down a list of all of the prime implicants from the map, note that there are often prime implicants which are not included in the minimum sum of products. Even though all of the 1's in a term have already been covered by prime Karnaugh Maps FIGURE 5-16 Determination of All Prime Implicants © Cengage Learning 2014 145 ab 00 cd 01 11 1 1 1 1 00 10 a′c′d 01 1 11 1 10 Minimum solution: F = a′b′d + bc′ + ac All prime implicants: a′b′d, bc′, ac, a′c′d, ab, b′cd 1 1 1 1 b′cd implicants, that term may still be a prime implicant provided that it is not included in a larger group of 1's. For example, in Figure 5-16, a′c′d is a prime implicant because it cannot be combined with other 1's to eliminate another variable. However, abd is not a prime implicant because it can be combined with two other 1's to form ab. The term b′cd is also a prime implicant even though both of its 1's are already covered by other prime implicants. In the process of finding prime implicants, don't-cares are treated just like 1's. However, a prime implicant composed entirely of don't-cares can never be part of the minimum solution. Because all of the prime implicants of a function are generally not needed in forming the minimum sum of products, a systematic procedure for selecting prime implicants is needed. If prime implicants are selected from the map in the wrong order, a nonminimum solution may result. For example, in Figure 5-17, if CD is chosen first, then BD, B′C, and AC are needed to cover the remaining 1's, and the solution contains four terms. However, if the prime implicants indicated in Figure 5-17(b) are chosen first, all 1's are covered and CD is not needed. In Section 6.2, prime implicant charts are defined. They can be used systematically to find (all) minimum solutions. The procedure described below can be used to find minimum solutions for functions that are not too complicated. Note that some of the minterms on the map of Figure 5-17(a) can be covered by only a single prime implicant, but other minterms can be covered by two different prime implicants. For example, m2 is covered only by B′C, but m3 is covered by both FIGURE 5-17 © Cengage Learning 2014 AB 00 CD 01 11 AB 00 CD 10 00 m5 01 11 1 10 1 01 11 10 1 1 1 1 1 1 1 00 1 1 1 1 1 11 1 1 1 1 10 1 01 CD m14 m2 f = CD + BD+ B′C + AC f = BD+ B′C + AC (a) (b) 146 Unit 5 B′C and CD. If a minterm is covered by only one prime implicant, that prime implicant is said to be essential, and it must be included in the minimum sum of products. Thus, B′C is an essential prime implicant because m2 is not covered by any other prime implicant. However, CD is not essential because each of the 1's in CD can be covered by another prime implicant. The only prime implicant which covers m5 is BD, so BD is essential. Similarly, AC is essential because no other prime implicant covers m14. In this example, if we choose all of the essential prime implicants, all of the 1's on the map are covered and the nonessential prime implicant CD is not needed. In general, in order to find a minimum sum of products from a map, we should first loop all of the essential prime implicants. One way of finding essential prime implicants on a map is simply to look at each 1 on the map that has not already been covered, and check to see how many prime implicants cover that 1. If there is only one prime implicant which covers the 1, that prime implicant is essential. If there are two or more prime implicants which cover the 1, we cannot say whether these prime implicants are essential or not without checking the other minterms. For simple problems, we can locate the essential prime implicants in this way by inspection of each 1 on the map. For example, in Figure 5-16, m4 is covered only by the prime implicant bc′, and m10 is covered only by the prime implicant ac. All other 1's on the map are covered by two prime implicants; therefore, the only essential prime implicants are bc′ and ac. For more complicated maps, and especially for maps with five or more variables, we need a more systematic approach for finding the essential prime implicants. When checking a minterm to see if it is covered by only one prime implicant, we must look at all squares adjacent to that minterm. If the given minterm and all of the 1's adjacent to it are covered by a single term, then that term is an essential prime implicant.[1] If all of the 1's adjacent to a given minterm are not covered by a single term, then there are two or more prime implicants which cover that minterm, and we cannot say whether these prime implicants are essential or not without checking the other minterms. Figure 5-18 illustrates this principle. FIGURE 5-18 © Cengage Learning 2014 AB CD 00 00 1 01 11 10 1 0 4 12 8 5 13 9 A′C′ 01 1 1 1 11 10 1 3 7 15 11 2 6 14 10 1 A′B′D′ 1 ACD 1 1 This statement is proved in Appendix D. Note: 1's shaded in blue are covered by only one prime implicant. All other 1's are covered by at least two prime implicants. Karnaugh Maps 147 The adjacent 1's for minterm m0 (l0) are 11, 12, and l4. Because no single term covers these four 1's, no essential prime implicant is yet apparent. The adjacent 1's for 11 are 10 and 15, so the term which covers these three 1's (A′C′) is an essential prime implicant. Because the only 1 adjacent to 12 is 10, A′B′D′ is also essential. Because the 1's adjacent to 17 (15 and 115) are not covered by a single term, neither A′BD nor BCD is essential at this point. However, because the only 1 adjacent to 111 is 115, ACD is essential. To complete the minimum solution, one of the nonessential prime implicants is needed. Either A′BD or BCD may be selected. The final solution is A′BD A′C′ + A′B′D′ + ACD + % or BCD If a don't-care minterm is present on the map, we do not have to check it to see if it is covered by one or more prime implicants. However, when checking a 1 for adjacent 1's, we treat the adjacent don't-cares as if they were 1's because don'tcares may be combined with 1's in the process of forming prime implicants. The following procedure can then be used to obtain a minimum sum of products from a Karnaugh map: 1. 2. 3. 4. 5. Choose a minterm (a 1) which has not yet been covered. Find all 1's and X's adjacent to that minterm. (Check the n adjacent squares on an n-variable map.) If a single term covers the minterm and all of the adjacent 1's and X's, then that term is an essential prime implicant, so select that term. (Note that don't-care terms are treated like 1's in steps 2 and 3 but not in step 1.) Repeat steps 1, 2, and 3 until all essential prime implicants have been chosen. Find a minimum set of prime implicants which cover the remaining 1's on the map. (If there is more than one such set, choose a set with a minimum number of literals.) Figure 5-19 gives a flowchart for this procedure. The following example (Figure 5-20) illustrates the procedure. Starting with 14, we see that the adjacent 1's and X's (X0, 15, and 16) are not covered by a single term, so no essential prime implicant is apparent. However, 16 and its adjacent 1's and X's (14 and X7) are covered by A′B, so A′B is an essential prime implicant. Next, looking at 113, we see that its adjacent 1's and X's (15, 19, and X15) are not covered by a single term, so no essential prime implicant is apparent. Similarly, an examination of the terms adjacent to 18 and 19 reveals no essential prime implicants. However, 110 has only 18 adjacent to it, so AB′D′ is an essential prime implicant because it covers both 110 and 18. Having first selected the essential prime implicants, we now choose AC′D because it covers both of the remaining 1's on the map. Judicious selection of the order in which the minterms are selected (step 1) reduces the amount of work required in applying this procedure. As will be seen in the next section, this procedure is especially helpful in obtaining minimum solutions for five- and six-variable problems. There are two equivalent methods of obtaining minimum product-of-sum expressions for a function f . As mentioned above, one method is to find minimum a sum-of-products expression for f ′, and then complement f ′ to obtain a minimum 148 Unit 5 FIGURE 5-19 Flowchart for Determining a Minimum Sum of Products Using a Karnaugh Map Choose a 1 which has not been covered. © Cengage Learning 2014 Find all adjacent 1's and X's. Are the chosen 1 and its adjacent 1's and X's covered by a single term? NO YES That term is an essential prime implicant. Loop it. All uncovered 1's checked? NO Note: All essential prime implicants have been determined at this point. YES Find a minimum set of prime implicants which cover the remaining 1's on the map. STOP FIGURE 5-20 © Cengage Learning 2014 AB CD 00 00 01 X0 14 11 10 18 01 15 113 11 X7 X15 10 16 19 110 Shaded 1's are covered by only one prime implicant. Karnaugh Maps 149 product-of-sums expression for f. Alternatively, we can perform the dual of the procedure for finding minimum sum of products. Let S be a sum term. If every input combination for which S = 0 f is also 0, then S can be a term in a product-of-sums expression for F. We will call such a sum term an implicate of f . Implicate S is a prime implicate if it cannot be combined with any other implicate to eliminate a literal from S. All implicates in a minimum product-of-sums expression for f must be prime implicates. The prime implicates of f can be found by looping the largest groups of adjacent zeros on the Karnaugh map for f . If a prime implicate is the only prime implicate covering a maxterm (zero) of f , then it is an essential prime implicate and must be included in any minimum product-of-sums expression for f . 5.5 Five-Variable Karnaugh Maps A five-variable map can be constructed in three dimensions by placing one four-variable map on top of a second one. Terms in the bottom layer are numbered 0 through 15 and corresponding terms in the top layer are numbered 16 through 31, so that terms in the bottom layer contain A′ and those in the top layer contain A. To represent the map in two dimensions, we will divide each square in a four-variable map by a diagonal line and place terms in the bottom layer below the line and terms in the top layer above the line (Figure 5-21). Terms in the top or bottom layer combine just like terms on a four-variable map. In addition, two terms in the same square which are separated by a diagonal line differ in only one variable and can be combined. However, some terms which appear to be physically adjacent are not. For example, terms 0 and 20 are not adjacent because they appear in a different column and a FIGURE 5-21 A Five-Variable Karnaugh Map These terms do not combine because they are in different layers and different columns (they differ in two variables). BC DE © Cengage Learning 2014 00 01 16 11 20 00 1 1 21 29 1 3 18 10 7 11 These four terms (two from top layer and two from bottom) combine to yield CDE(C from the middle two columns and DE from the row). 27 1 22 1 1 13 1 9 These eight terms combine to give BD′(B from last two columns and D′ from top two rows; A is eliminated because four terms are in the top layer and four in the bottom). 1 1 31 1 11 8 25 5 23 1 12 1 19 1 1 4 01 A 1 0 24 1 0 17 10 28 1 15 30 26 1 2 6 14 10 These two terms in the top layer combine to give AB′DE′. 150 Unit 5 different layer. Each term can be adjacent to exactly five other terms, four in the same layer and one in the other layer (Figure 5-22). An alternate representation for five-variable maps is to draw the two layers side-by-side, as in Figure 5-28, but most individuals find adjacencies more difficult to see when this form is used. When checking for adjacencies, each term should be checked against the five possible adjacent squares. (In general, the number of adjacent squares is equal to the number of variables.) Two examples of five-variable minimization using maps follow. Figure 5-23 is a map of F(A, B, C, D, E) = Σ m(0, 1, 4, 5, 13, 15, 20, 21, 22, 23, 24, 26, 28, 30, 31) FIGURE 5-22 © Cengage Learning 2014 BC 00 DE 01 00 11 10 1 1 01 A 1 0 1 1 11 1 1 10 FIGURE 5-23 BC DE © Cengage Learning 2014 00 01 1 00 1 1 11 10 1 1 24 Shaded 1's are used to select essential prime implicants. 0 P1 01 A 1 0 1 1 1 1 11 1 1 1 10 1 1 P3 P4 1 1 P2 Karnaugh Maps 151 Prime implicant P1 is chosen first because all of the 1's adjacent to minterm 0 are covered by P1. Prime implicant P2 is chosen next because all of the 1's adjacent to minterm 24 are covered by P2. All of the remaining 1's on the map can be covered by at least two different prime implicants, so we proceed by trial and error. After a few tries, it becomes apparent that the remaining 1's can be covered by three prime implicants. If we choose prime implicants P3 and P4 next, the remaining two 1's can be covered by two different groups of four. The resulting minimum solution is AB′C F = A′B′D′ + ABE′ + ACD + A′BCE + % or P1 P2 P3 P4 B′CD′ Figure 5-24 is a map of F (A, B, C, D, E) = Σ m(0, 1, 3, 8, 9, 14, 15, 16, 17, 19, 25, 27, 31) All 1's adjacent to m0 are covered by P1, so choose P1 first. All 1's adjacent to m3 are covered by P2, so P2 is chosen next. All 1's adjacent to m8 are covered by P3, so P3 is chosen. Because m14 is only adjacent to m15, P4 is also essential. There are no more essential prime implicants, and the remaining 1's can be covered by two terms, P5 and (1-9-17-25) or (17-19-25-27). The final solution is C′D′E F = B′C′D′ + B′C′E + A′C′D′ + A′BCD + ABDE + % or P1 P2 P3 P4 P5 AC′E FIGURE 5-24 BC DE © Cengage Learning 2014 16 P1 00 01 1 00 10 28 P3 24 1 1 4 0 17 21 1 01 A 1 0 11 20 5 23 1 8 25 1 19 P2 29 1 1 11 12 13 31 9 27 1 1 1 P5 1 1 3 18 7 22 15 30 10 11 26 1 2 6 14 P4 10 152 Unit 5 5.6 Other Uses of Karnaugh Maps Many operations that can be performed using a truth table or algebraically can be done using a Karnaugh map. A map conveys the same information as a truth table— it is just arranged in a different format. If we plot an expression for F on a map, we can read off the minterm and maxterm expansions for F and for F ′. From the map of Figure 5-14, the minterm expansion of f is f = Σ m(0, 2, 3, 4, 8, 10, 11, 15) and because each 0 corresponds to a maxterm, the maxterm expansion of f is f = Π M(1, 5, 6, 7, 9, 12, 13, 14) We can prove that two functions are equal by plotting them on maps and showing that they have the same Karnaugh map. We can perform the AND operation (or the OR operation) on two functions by ANDing (or ORing) the 1's and 0's which appear in corresponding positions on their maps. This procedure is valid because it is equivalent to doing the same operations on the truth tables for the functions. A Karnaugh map can facilitate factoring an expression. Inspection of the map reveals terms which have one or more variables in common. For the map of Figure 5-25, the two terms in the first column have A′B′ in common; the two terms in the lower right corner have AC in common. FIGURE 5-25 © Cengage Learning 2014 AB CD 00 00 1 01 1 11 1 01 11 10 F = A′B′(C′ + D) + AC(B + D′) 10 1 1 1 When simplifying a function algebraically, the Karnaugh map can be used as a guide in determining what steps to take. For example, consider the function F = ABCD + B′CDE + A′B′ + BCE′ From the map (Figure 5-26), we see that in order to get the minimum solution, we must add the term ACDE. We can do this using the consensus theorem: F = ABCD + B′CDE + A′B′ + BCE′ + ACDE ↑ Karnaugh Maps 153 As can be seen from the map, this expression now contains two redundant terms, ABCD and B′CDE. These can be eliminated using the consensus theorem, which gives the minimum solution: F = A′B + BCE′ + ACDE FIGURE 5-26 BC 00 DE © Cengage Learning 2014 01 16 00 1 01 21 12 29 8 25 1 1 19 5 23 13 31 1 7 22 1 15 30 6 11 Add this term. . 26 1 1 2 9 27 1 1 3 18 10 1 4 1 11 24 1 1 17 10 28 0 A 1 0 11 20 1 14 10 Then these two terms can be eliminated. 5.7 Other Forms of Karnaugh Maps Instead of labeling the sides of a Karnaugh map with 0's and 1's, some people prefer to use the labeling shown in Figure 5-27. For the half of the map labeled A, A = 1; and for the other half, A = 0. The other variables have a similar interpretation. A map labeled this way is sometimes referred to as a Veitch diagram. It is particularly useful for plotting functions given in algebraic form rather than in minterm or maxterm form. However, when utilizing Karnaugh maps to solve sequential circuit problems (Units 12 through 16), the use of 0's and 1's to label the maps is more convenient. FIGURE 5-27 Veitch Diagrams A A © Cengage Learning 2014 C B D C B 154 Unit 5 Two alternative forms for five-variable maps are used. One form simply consists of two four-variable maps side-by-side as in Figure 5-28(a). A modification of this uses a mirror image map as in Figure 5-28(b). In this map, first and eighth columns are “adjacent” as are second and seventh columns, third and sixth columns, and fourth and fifth columns. The same function is plotted on both these maps. FIGURE 5-28 Other Forms of Five-Variable Karnaugh Maps © Cengage Learning 2014 A BC DE 00 00 1 01 1 01 11 10 1 1 BC DE 00 00 1 1 01 1 11 01 11 10 1 1 1 1 B 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 E 11 10 1 1 1 D 10 A= 0 1 1 A= 1 C C (a) (b) F = D′E′ + B′C′D′ + BCE + A′BC′E′ + ACDE Programmed Exercise 5.1 Cover the answers to this exercise with a sheet of paper and slide it down as you check your answers. Write your answers in the space provided before looking at the correct answer. Problem sums for Determine the minimum sum of products and minimum product of f = b′c′d′ + bcd + acd′ + a′b′c + a′bc′d First, plot the map for f . 00 00 01 11 10 01 11 10 Karnaugh Maps Answer: 155 ab 00 cd 01 11 10 1 00 1 1 01 11 1 10 1 1 1 1 1 1 (a) The minterms adjacent to m0 on the preceding map are _________ and _________. (b) Find an essential prime implicant containing m0 and loop it. (c) The minterms adjacent to m3 are _________ and _________. (d) Is there an essential prime implicant which contains m3? (e) Find the remaining essential prime implicant(s) and loop it (them). Answers: ab (a) m2 and m8 (c) m2 and m7 (d) No (b) (e) 00 cd 00 01 1 1 11 1 10 1 1 01 11 1 10 1 1 1 1 1 Loop the remaining 1's using a minimum number of loops. The two possible minimum sum-of-products forms for f are f = ______________________________ and f = ______________________________ Answer: ab cd 00 00 01 11 1 10 1 1 01 f = b′d′ + a′bd + abc + 11 1 10 1 1 1 1 1 a′cd or a′b′c 156 Unit 5 Next, we will find the minimum product of sums for f . Start by plotting the map for f ′. Loop all essential prime implicants of f ′ and indicate which minterm makes each one essential. 00 01 11 10 00 01 11 10 f Answer: ab cd 00 00 01 01 11 1 1 1 1 10 1 Essential because of m1 1 11 10 Essential because of m11 Essential because of m6 1 f Loop the remaining 1's and write the minimum sum of products for f ′. f ′= ______________________________ The minimum product of sums for f is therefore f = ______________________________ Final Answer: f ′ = b′c′d + a′bd′ + ab′d + abc′ f = (b + c + d′)(a + b′ + d)(a′ + b + d′)(a′ + b′ + c) Programmed Exercise 5.2 Problem: Determine a minimum sum-of-products expression for f (a, b, c, d, e) = (a′ + c + d)(a′ + b + e) (a + c′ + e′) (c + d + e′) (b + c + d′ + e) (a′ + b′ + c + e′) Karnaugh Maps 157 The first step in the solution is to plot a map for f . Because f is given in productof-sums form, it is easier to first plot the map for f ′ and then complement the map. Write f ′ as a sum of products: f ′ = ______________________________ Now plot the map for f ′. (Note that there are three terms in the upper layer, one term in the lower layer, and two terms which span the two layers.) Next, convert your map for f ′ to a map for f . bc bc 00 de 01 11 10 00 de 00 01 11 10 00 01 01 a 1 0 a 1 0 11 11 10 10 f′ f Answer: bc de bc 00 01 11 10 00 de 01 16 00 1 1 1 11 20 1 1 17 01 1 1 1 1 11 1 1 1 1 1 a 1 0 29 1 1 5 31 1 1 1 1 1 7 15 22 30 1 2 1 1 6 1 14 f 11 26 1 10 f′ 9 27 1 18 1 13 23 1 11 8 25 1 3 10 12 21 19 1 1 4 01 1 24 1 00 0 a 1 0 10 28 10 158 Unit 5 The next step is to determine the essential prime implicants of f . (a) Why is a′d′e′ an essential prime implicant? (b) Which minterms are adjacent to m3? __________ To m19? __________ (c) Is there an essential prime implicant which covers m3 and m19? (d) Is there an essential prime implicant which covers m21? (e) Loop the essential prime implicants which you have found. Then, find two more essential prime implicants and loop them. Answers: (a) (b) (c) (d) (e) It covers m0 and both adjacent minterms. m19 and m11; m3 and m23 No Yes bc de 00 01 11 10 1 00 1 01 1 1 1 1 1 1 1 a 1 0 11 1 1 1 1 1 1 1 1 10 1 1 1 (a) Why is there no essential prime implicant which covers m11? (b) Why is there no essential prime implicant which covers m28? Because there are no more essential prime implicants, loop a minimum number of terms which cover the remaining 1's. Answers: (a) All adjacent 1's of m11(m3, m10) cannot be covered by one grouping. (b) All adjacent 1's of m28(m12, m30, m29) cannot be covered by one grouping. Karnaugh Maps 159 bc de 00 01 1 1 1 11 10 1 00 1 01 a 1 0 1 11 1 1 1 1 Note: There are five other possible ways to loop the four remaining 1's. 1 1 1 10 1 1 1 1 1 Write down two different minimum sum-of-products expressions for f . f = ______________________________ f = ______________________________ Answer: abc b′c′de + a′c′de f = a′d′e′ + ace + a′ce′ + bde′ + | or ¶ + | b′c′de + a′ bc′d ¶ bce′ ab′de + a′c′de Problems 5.3 Find the minimum sum of products for each function using a Karnaugh map. (a) f1(a, b, c) = m0 + m2 + m5 + m6 (b) f2(d, e, f) = Σ m(0, 1, 2, 4) (c) f3(r, s, t) = rt ′ + r′s′ + r′s (d) f4(x, y, z) = M0 · M5 5.4 (a) Plot the following function on a Karnaugh map. (Do not expand to minterm form before plotting.) F(A, B, C, D) = BD′ + B′CD + ABC + ABC′D + B′D′ (b) Find the minimum sum of products. (c) Find the minimum product of sums. 160 Unit 5 5.5 A switching circuit has two control inputs (C1 and C2), two data inputs (X1 and X2), and one output (Z). The circuit performs one of the logic operations AND, OR, EQU (equivalence), or XOR (exclusive OR) on the two data inputs. The function performed depends on the control inputs: C1 0 0 1 1 C2 0 1 0 1 Function Performed by Circuit OR XOR AND EQU (a) Derive a truth table for Z. (b) Use a Karnaugh map to find a minimum AND-OR gate circuit to realize Z. 5.6 Find the minimum sum-of-products expression for each function. Underline the essential prime implicants in your answer and tell which minterm makes each one essential. (a) f(a, b, c, d) = Σ m(0, 1, 3, 5, 6, 7, 11, 12, 14) (b) f(a, b, c, d) = Π M(1, 9, 11, 12, 14) (c) f(a, b, c, d) = Π M(5, 7, 13, 14, 15) · Π D(1, 2, 3, 9) 5.7 Find the minimum sum-of-products expression for each function. (a) f(a, b, c, d) = Σ m(0, 2, 3, 4, 7, 8, 14) (b) f(a, b, c, d) = Σ m(1, 2, 4, 15) + Σ d(0, 3, 14) (c) f(a, b, c, d) = Π M(1, 2, 3, 4, 9, 15) (d) f(a, b, c, d) = Π M(0, 2, 4, 6, 8) · Π D(1, 12, 9, 15) 5.8 Find the minimum sum of products and the minimum product of sums for each function: (a) f(a, b, c, d) = Π M(0, 1, 6, 8, 11, 12) · Π D(3, 7, 14, 15) (b) f(a, b, c, d) = Σ m(1, 3, 4, 11) + Σ d(2, 7, 8, 12, 14, 15) 5.9 Find the minimum sum of products and the minimum product of sums for each function: (a) F(A, B, C, D, E) = Σ m(0, 1, 2, 6, 7, 9, 10, 15, 16, 18, 20, 21, 27, 30) + Σ d(3, 4, 11, 12, 19) (b) F(A, B, C, D, E) = Π M(0, 3, 6, 9, 11, 19, 20, 24, 25, 26, 27, 28, 29, 30) · Π D(1, 2, 12, 13) 5.10 F(a, b, c, d, e) = Σ m(0, 3, 4, 5, 6, 7, 8, 12, 13, 14, 16, 21, 23, 24, 29, 31) (a) Find the essential prime implicants using a Karnaugh map, and indicate why each one of the chosen prime implicants is essential (there are four essential prime implicants). (b) Find all of the prime implicants by using the Karnaugh map. (There are nine in all.) Karnaugh Maps 161 5.11 Find a minimum product-of-sums solution for f . Underline the essential prime implicates. f(a, b, c, d, e) = Σ m(2, 4, 5, 6, 7, 8, 10, 12, 14, 16, 19, 27, 28, 29, 31) + Σ d(1, 30) 5.12 Given F = AB′D′ + A′B + A′C + CD. (a) Use a Karnaugh map to find the maxterm expression for F (express your answer in both decimal and algebraic notation). (b) Use a Karnaugh map to find the minimum sum-of-products form for F ′. (c) Find the minimum product of sums for F. 5.13 Find the minimum sum of products for the given expression. Then, make minterm 5 a don't-care term and verify that the minimum sum of products is unchanged. Now, start again with the original expression and find each minterm which could individually be made a don't-care without changing the minimum sum of products. F(A, B, C, D) = A′C′ + B′C + ACD′ + BC′D 5.14 Find the minimum sum-of-products expressions for each of these functions. (a) f1(A, B, C) = m1 + m2 + m5 + m7 (b) f2(d, e, f) = Σ m(1, 5, 6, 7) (c) f3(r, s, t) = rs′ + r′s′ + st ′ (d) f4(a, b, c) = m0 + m2 + m3 + m7 (e) f5(n, p, q) = Σ m(1, 3, 4, 5) (f) f6(x, y, z) = M1M7 5.15 Find the minimum product-of-sums expression for each of the functions in Problem 5.14. 5.16 Find the minimum sum of products for each of these functions. (a) f1(A, B, C) = m1 + m3 + m4 + m6 (b) f2(d, e, f) = Σ m(1, 4, 5, 7) (c) f3(r, s, t) = r ′t ′ + rs′ + rs (d) f1(a, b, c) = m3 + m4 + m6 + m7 (e) f2(n, p, q) = Σ m(2, 3, 5, 7) (f) f4(x, y, z) = M3M6 5.17 (a) Plot the following function on a Karnaugh map. (Do not expand to minterm form before plotting.) F(A, B, C, D) = A′B′ + CD′ + ABC + A′B′CD′ + ABCD′ (b) Find the minimum sum of products. (c) Find the minimum product of sums. 5.18 Work Problem 5.17 for the following: f(A, B, C, D) = A′B′ + A′BC′ + A′BD′ + AC′D + A′BD + AB′CD′ 162 Unit 5 5.19 A switching circuit has two control inputs (C1 and C2), two data inputs (X1 and X2), and one output (Z). The circuit performs logic operations on the two data inputs, as shown in this table: C1 0 0 1 1 C2 0 1 0 1 Function Performed by Circuit X1X2 X1 ⊕ X2 X′1 + X2 X 1 ≡ X2 (a) Derive a truth table for Z. (b) Use a Karnaugh map to find a minimum OR-AND gate circuit to realize Z. 5.20 Use Karnaugh maps to find all possible minimum sum-of-products expressions for each function. (a) F(a, b, c) = Π M(3, 4) (b) g(d, e, f) = Σ m(1, 4, 6) + Σ d(0, 2, 7) (c) F(p, q, r) = (p + q′ + r)(p′ + q + r′) (d) F(s, t, u) = Σ m(1, 2, 3) + Σ d(0, 5, 7) (e) f(a, b, c) = Π M(2, 3, 4) (f) G(D, E, F) = Σ m(1, 6) + Σ d(0, 3, 5) 5.21 Simplify the following expression first by using a map and then by using Boolean algebra. Use the map as a guide to determine which theorems to apply to which terms for the algebraic simplification. F = a′b′c′ + a′c′d + bcd + abc + ab′ 5.22 Find all prime implicants and all minimum sum-of-products expressions for each of the following functions. (a) f(A, B, C, D) = Σ m(4, 11, 12, 13, 14) + Σ d(5, 6, 7, 8, 9, 10) (b) f(A, B, C, D) = Σ m(3, 11, 12, 13, 14) + Σ d(5, 6, 7, 8, 9, 10) (c) f(A, B, C, D) = Σ m(1, 2, 4, 13, 14) + Σ d(5, 6, 7, 8, 9, 10) (d) f(A, B, C, D) = Σ m(4, 15) + Σ d(5, 6, 7, 8, 9, 10) (e) f(A, B, C, D) = Σ m(3, 4, 11, 15) + Σ d(5, 6, 7, 8, 9, 10) (f) f(A, B, C, D) = Σ m(4) + Σ d(5, 6, 7, 8, 9, 10, 11, 12, 13, 14) (g) f(A, B, C, D) = Σ m(4, 15) + Σ d(0, 1, 2, 5, 6, 7, 8, 9, 10) 5.23 For each function in Problem 5.22, find all minimum product-of-sums expressions. 5.24 Find the minimum sum-of-products expression for (a) Σ m(0, 2, 3, 5, 6, 7, 11, 12, 13) (b) Σ m(2, 4, 8) + Σ d(0, 3, 7) (c) Σ m(1, 5, 6, 7, 13) + Σ d(4, 8) (d) f(w, x, y, z) = Σ m(0, 3, 5, 7, 8, 9, 10, 12, 13) + Σ d(1, 6, 11, 14) (e) Π M(0, 1, 2, 5, 7, 9, 11) · Π D(4, 10, 13) Karnaugh Maps 163 5.25 Work Problem 5.24 for the following: (a) f (a, b, c, d) = Σ m(1, 3, 4, 5, 7, 9, 13, 15) (b) f (a, b, c, d) = Π M(0, 3, 5, 8, 11) (c) f (a, b, c, d) = Σ m(0, 2, 6, 9, 13, 14) + Σ d(3, 8, 10) (d) f (a, b, c, d) = Π M(0, 2, 6, 7, 9, 12, 13) · Π D(1, 3, 5) 5.26 Find the minimum product of sums for the following. Underline the essential prime implicates in your answer. (a) Π M(0, 2, 4, 5, 6, 9, 14) · Π D(10, 11) (b) Σ m(1, 3, 8, 9, 15) + Σ d(6, 7, 12) 5.27 Find a minimum sum-of-products and a minimum product-of-sums expression for each function: (a) f (A, B, C, D) = Π M(0, 2, 10, 11, 12, 14, 15) · Π D(5, 7) (b) f (w, x, y, z) = Σ m(0, 3, 5, 7, 8, 9, 10, 12, 13) + Σ d(1, 6, 11, 14) 5.28 A logic circuit realizes the function F(a, b, c, d) = a′b′ + a′cd + ac′d + ab′d′. Assuming that a = c never occurs when b = d = 1, find a simplified expression for F. 5.29 Given F = AB′D′ + A′B + A′C + CD. (a) Use a Karnaugh map to find the maxterm expression for F (express your answer in both decimal and algebraic notation). (b) Use a Karnaugh map to find the minimum sum-of-products form for F ′. (c) Find the minimum product of sums for F. 5.30 Assuming that the inputs ABCD = 0101, BCD = 1001, ABCD = 1011 never occur, find a simplified expression for F = A′BC′D + A′B′D + A′CD + ABD + ABC 5.31 Find all of the prime implicants for each of the functions plotted on page 157. 5.32 Find all of the prime implicants for each of the plotted functions: bc bc 00 de 01 1 11 10 00 1 1 1 a 1 0 10 1 1 1 1 1 1 1 1 1 01 11 10 1 1 00 01 00 de 1 a 1 0 11 F 1 1 01 10 1 1 1 1 1 1 1 1 1 1 1 G 1 164 Unit 5 5.33 Given that f (a, b, c, d, e) = Σ m(6, 7, 9, 11, 12, 13, 16, 17, 18, 20, 21, 23, 25, 28), using a Karnaugh map, (a) Find the essential prime implicants (three). (b) Find the minimum sum of products (7 terms). (c) Find all of the prime implicants (twelve). 5.34 A logic circuit realizing the function f has four inputs a, b, c, d. The three inputs a, b, and c are the binary representation of the digits 0 through 7 with a being the most significant bit. The input d is an odd-parity bit; that is, the value of d is such that a, b, c, and d always contains an odd number of 1's. (For example, the digit 1 is represented by abc = 001 and d = 0, and the digit 3 is represented by abcd = 0111.) The function f has value 1 if the input digit is a prime number. (A number is prime if it is divisible only by itself and 1; 1 is considered to be prime, and 0 is not.) (a) Draw a Karnaugh map for f . (b) Find all prime implicants of f . (c) Find all minimum sum of products for f . (d) Find all prime implicants of f ′. (e) Find all minimum product of sums for f . 5.35 The decimal digits 0 though 9 are represented using five bits A, B, C, D, and E. The bits A, B, C, and D are the BCD representation of the decimal digit, and bit E is a parity bit that makes the five bits have odd parity. The function F (A, B, C, D, E) has value 1 if the decimal digit represented by A, B, C, D, and E is divisible by either 3 or 4. (Zero is divisible by 3 and 4.) (a) Draw a Karnaugh map for f . (b) Find all prime implicants of f . (Prime implicants containing only don't-cares need not be included.) (c) Find all minimum sum of products for f . (d) Find all prime implicants of f ′. (e) Find all minimum product of sums for f . 5.36 Rework Problem 5.35 assuming the decimal digits are represented in excess-3 rather than BCD. 5.37 The function F(A, B, C, D, E) = Σ m(1, 7, 8, 13, 16, 19) + Σ d(0, 3, 5, 6, 9, 10, 12, 15, 17, 18, 20, 23, 24, 27, 29, 30). (a) Draw a Karnaugh map for f . (b) Find all prime implicants of f . (Prime implicants containing only don't-cares need not be included.) (c) Find all minimum sum of products for f . (d) Find all prime implicants of f ′. (e) Find all minimum product of sums for f . Karnaugh Maps 165 5.38 F(a, b, c, d, e) = Σ m(0, 1, 4, 5, 9, 10, 11, 12, 14, 18, 20, 21, 22, 25, 26, 28) (a) Find the essential prime implicants using a Karnaugh map, and indicate why each one of the chosen prime implicants is essential (there are four essential prime implicants). (b) Find all of the prime implicants by using the Karnaugh map (there are 13 in all). 5.39 Find the minimum sum-of-products expression for F. Underline the essential prime implicants in this expression. (a) f(a, b, c, d, e) = Σ m(0, 1, 3, 4, 6, 7, 8, 10, 11, 15, 16, 18, 19, 24, 25, 28, 29, 31) + Σ d(5, 9, 30) (b) f(a, b, c, d, e) = Σ m(1, 3, 5, 8, 9, 15, 16, 20, 21, 23, 27, 28, 31) 5.40 Work Problem 5.39 with F(A, B, C, D, E) = Π M(2, 3, 4, 8, 9, 10, 14, 15, 16, 18, 19, 20, 23, 24, 30, 31) 5.41 Find the minimum sum-of-products expression for F. Underline the essential prime implicants in your expression. F(A, B, C, D, E) = Σ m(0, 2, 3, 5, 8, 11, 13, 20, 25, 26, 30) + Σ d(6, 7, 9, 24) 5.42 F(V, W, X, Y, Z) = Π M(0, 3, 5, 6, 7, 8, 11, 13, 14, 15, 18, 20, 22, 24) · Π D(1, 2, 16, 17) (a) Find a minimum sum-of-products expression for F. Underline the essential prime implicants. (b) Find a minimum product-of-sums expression for F. Underline the essential prime implicates. 5.43 Find the minimum product of sums for (a) F(a, b, c, d, e) = Σ m(1, 2, 3, 4, 5, 6, 25, 26, 27, 28, 29, 30, 31) (b) F(a, b, c, d, e) = Σ m(1, 5, 12, 13, 14, 16, 17, 21, 23, 24, 30, 31) + Σ d(0, 2, 3, 4) 5.44 Find a minimum product-of-sums expression for each of the following functions: (a) F(v, w, x, y, z) = Σ m(4, 5, 8, 9, 12, 13, 18, 20, 21, 22, 25, 28, 30, 31) (b) F(a, b, c, d, e) = Π M(2, 4, 5, 6, 8, 10, 12, 13, 16, 17, 18, 22, 23, 24) · Π D(0, 11, 30, 31) 5.45 Find the minimum sum of products for each function. Then, make the specified minterm a don't-care and verify that the

minimum sum of products is unchanged. Now, start again with the original expression and find each minterm which could individually be made a don't-care, without changing the minimum sum of products. (a) F(A, B, C, D) = A′C′ + A′B′ + ACD′ + BC′D, minterm 2 (b) F(A, B, C, D) = A′BD + AC′D + AB′ + BCD + A′C′D, minterm 7 166 Unit 5 5.46 F(V, W, X, Y, Z) = Π M(0, 3, 6, 9, 11, 19, 20, 24, 25, 26, 27, 28, 29, 30) · Π D(1, 2, 12, 13) (a) Find two minimum sum-of-products expressions for F. (b) Underline the essential prime implicants in your answer and tell why each one is essential. 5.47 Four of the minterms of the completely specified function f(a, b, c, d) are m0, m1, m4, and m5. (a) Specify additional minterms for f so that f has eight prime implicants with two literals and no other prime implicants. (b) For each prime implicant, give its algebraic representation and specify whether it is an essential prime implicant. (c) Determine all minimum sum-of-products expressions for f . 5.48 Four of the minterms of the completely specified function f(a, b, c, d) are m0, m1, m4, and m5. (a) Specify additional minterms for f so that f has one prime implicant with one literal, six prime implicants with two literals, and no other prime implicants. (b) For each prime implicant, give its algebraic representation and specify whether it is an essential prime implicant. (c) Determine all minimum sum-of-products expressions for f . 5.49 Four of the minterms of the completely specified function f(a, b, c, d) are m0, m1, m4, and m5. (a) Specify additional minterms for f so that f has two prime implicants with one literal, two prime implicants with two literals, and no other prime implicants. (b) For each prime implicant, give its algebraic representation and specify whether it is an essential prime implicant. (c) Determine all minimum sum-of-products expressions for f . 5.50 Four of the minterms of an incompletely specified function f(a, b, c, d) are m0, m1, m4, and m5. (a) Specify additional minterms and don't-cares for f so that f has five prime implicants with two literals and no other prime implicants and, in addition, f has one prime implicate with one literal and two prime implicates with two literals. (b) For each prime implicant, give its algebraic representation and specify whether it is an essential prime implicant. (c) Determine all minimum sum-of-products expressions for f . (d) For each prime implicate, give its algebraic representation and specify whether it is an essential prime implicate. (e) Determine all minimum product-of-sums expressions for f . UNIT Quine-McCluskey Method 6 Objectives 1. Find the prime implicants of a function by using the Quine-McCluskey method. Explain the reasons for the procedures used. 2. Define prime implicant and essential prime implicant. 3. Given the prime implicants, find the essential prime implicants and a minimum sum-of-products expression for a function, using a prime implicant chart and using Petrick's method. 4. Minimize an incompletely specified function, using the Quine-McCluskey method. 5. Find a minimum sum-of-products expression for a function, using the method of map-entered variables. 167 168 Unit 6 Study Guide 1. Review Section 5.1, Minimum Forms of Switching Functions. 2. Read the introduction to this unit and, then, study Section 6.1. Determination of Prime Implicants. (a) Using variables A, B, C, D, and E, give the algebraic equivalent of 10110 + 10010 = 10−10 10−10 + 10−11 = 10−1− (b) Why will the following pairs of terms not combine? 01101 + 00111 10−10 + 001−0 (c) When using the Quine-McCluskey method for finding prime implicants, why is it necessary to compare terms only from adjacent groups? (d) How can you determine if two minterms from adjacent groups will combine by looking at their decimal representations? (e) When combining terms, why is it permissible to use a term which has already been checked off? (f) In forming Column II of Table 6-1, note that terms 10 and 14 were combined to form 10, 14 even though both 10 and 14 had already been checked off. If this had not been done, which term in Column II could not be eliminated (checked off)? (g) In forming Column III of Table 6-1, note that minterms 0, 1, 8, and 9 were combined in two different ways to form –00–. This is equivalent to looping the minterms in two different ways on the Karnaugh map, as shown. ab cd ab 00 01 11 10 00 1 1 01 1 1 cd = ab 00 01 11 10 00 1 1 01 1 1 cd = 11 10 1 1 01 1 1 11 11 10 10 10 (0, 8) + (1, 9) 01 00 11 (0, 1) + (8, 9) 00 (0, 1, 8, 9) Quine-McCluskey Method 169 (h) Using a map, find all of the prime implicants of Equation (6-2) and compare your answer with Equation (6-3). 00 01 11 10 00 01 11 10 00 01 11 10 (i) The prime implicants of f(a, b, c, d) = Σ m(4, 5, 6, 7, 12, 13, 14, 15) are to be found using the Quine-McCluskey method. Column III is given; find Column IV and check off the appropriate terms in Column III. (4, 5, 6, 7) (4, 5, 12, 13) (4, 6, 12, 14) (5, 7, 13, 15) (6, 7, 14, 15) (12, 13, 14, 15) Column III 01 - –10– –1–0 –1–1 –11– 11 - - Column IV 00 01 11 10 00 01 11 10 Check your answer using a Karnaugh map. 3. (a) List all seven product term implicants of F(a, b, c) = Σ m(0, 1, 5, 7) Which of these implicants are prime? Why is a′c not an implicant? (b) Define a prime implicant. (c) Why must every term in a minimum sum-of-products expression be a prime implicant? 170 Unit 6 (d) Given that F(A, B, C, D) = Σ m(0, 1, 4, 5, 7, 10, 15), which of the following terms are not prime implicants and why? A′B′C′ 4. A′C′ BCD ABC AB′CD′ Study Section 6.2, The Prime Implicant Chart. (a) Define an essential prime implicant. (b) Find all of the essential prime implicants from the following chart. (0, 4) (4, 5, 12, 13) (13, 15) (11, 15) (10, 11) a 0 – 1 1 1 b – 1 1 – 0 c 0 0 – 1 1 d 0 – 1 1 – 0 4 5 10 11 12 13 15 ×× × × × × × × × × × Check your answer using a Karnaugh map. (c) Why must all essential prime implicants of a function be included in the minimum sum of products? (d) Complete the solution of Table 6-5. (e) Work Programmed Exercise 6.1. (f) Work Problems 6.2 and 6.3. 5. Study Section 6.3, Petrick's Method (optional). (a) Consider the following reduced prime implicant chart for a function F: m4 m5 m7 m13 P1 P2 P3 P4 bd bc′ a′b c′d × × × × × × × × × × × × We will find all minimum solutions using Petrick's method. Let Pi = 1 mean the prime implicant in row Pi is included in the solution. Which minterm is covered iff (P1 + P3) = 1?________ Write a sum term which is 1 iff m4 is covered.________ Quine-McCluskey Method 171 Write a product-of-sum terms which is 1 iff all m4, m5, m7 and m13 are all covered: P = ________________ (b) Reduce P to a minimum sum of products. (Your answer should have four terms, each one of the form Pi Pj.) P = ________________ If P1P2 = 1, which prime implicants are included in the solution?________ How many minimum solutions are there?________ Write out each solution in terms of a, b, c, and d. 6. (1) F = (2) F = (3) F = (4) F = Study Section 6.4, Simplification of Incompletely Specified Functions. (a) Why are don't-care terms treated like required minterms when finding the prime implicants? (b) Why are the don't-care terms not listed at the top of the prime implicant chart when finding the minimum solution? (c) Work Problem 6.4. (d) Work Problem 6.5, and check your solution using a Karnaugh map. 7. If you have LogicAid or a similar computer program available, use it to check your answers to some of the problems in this unit. LogicAid accepts Boolean functions in the form of equations, minterms or maxterms, and truth tables. It finds simplified sum-of-products and product-of-sums expressions for the functions using a modified version of the Quine-McCluskey method or Espresso-II. It can also find one or all of the minimum solutions using Petrick's method. 8. Study Section 6.5, Simplification Using Map-Entered Variables. (a) For the following map, find MS0, MS1, and F. Verify that your solution for F is minimum by using a four-variable map. A 0 1 D 1 11 1 D 10 1 X BC 00 01 172 Unit 6 (b) Use the method of map-entered variables to find an expression for F from the following map. Treat C and C′ as if they were independent variables. Is the result a correct representation of F? Is it minimum? A 0 B 0 1 1 C C′ 1 (c) Work Problem 6.6. 9. In this unit you have learned a "turn-the-crank" type procedure for finding minimum sum-of-products forms for switching functions. In addition to learning how to "turn the crank" and grind out minimum solutions, you should have learned several very important concepts in this unit. In particular, make sure you know: (a) (b) (c) (d) 10. What a prime implicant is What an essential prime implicant is Why the minimum sum-of-products form is a sum of prime implicants How don't-cares are handled when using the Quine-McCluskey method and the prime implicant chart Reread the objectives of the unit. If you are satisfied that you can meet the objectives, take the readiness test. Quine-McCluskey Method The Karnaugh map method described in Unit 5 is an effective way to simplify switching functions which have a small number of variables. When the number of variables is large or if several functions must be simplified, the use of a digital computer is desirable. The Quine-McCluskey method presented in this unit provides a systematic simplification procedure which can be readily programmed for a digital computer. Quine-McCluskey Method 173 The Quine-McCluskey method reduces the minterm expansion (standard sumof-products form) of a function to obtain a minimum sum of products. The procedure consists of two main steps: 1. 2. Eliminate as many literals as possible from each term by systematically applying the theorem XY + XY′ = X. The resulting terms are called prime implicants. Use a prime implicant chart to select a minimum set of prime implicants which, when ORed together, are equal to the function being simplified and which contain a minimum number of literals. 6.1 Determination of Prime Implicants In order to apply the Quine-McCluskey method to determine a minimum sum-ofproducts expression for a function, the function must be given as a sum of minterms. (If the function is not in minterm form, the minterm expansion can be found by using one of the techniques given in Section 5.3.) In the first part of the QuineMcCluskey method, all of the prime implicants of a function are systematically formed by combining minterms. The minterms are represented in binary notation and combined using XY + XY′ = X (6-1) where X represents a product of literals and Y is a single variable. Two minterms will combine if they differ in exactly one variable. In order to find all of the prime implicants, all possible pairs of minterms should be compared and combined whenever possible. To reduce the required number of comparisons, the binary minterms are sorted into groups according to the number of 1's in each term. Thus, f(a, b, c, d) = Σ m(0, 1, 2, 5, 6, 7, 8, 9, 10, 14) is represented by the following list of minterms: group 0 0 0000 group 1 | 1 0001 2 0010 8 1000 5 6 group 2 & 9 10 7 group 3 e 14 0101 0110 1001 1010 0111 1110 (6-2) 174 Unit 6 In this list, the term in group 0 has zero 1's, the terms in group 1 have one 1, those in group 2 have two 1's, and those in group 3 have three 1's. Two terms can be combined if they differ in exactly one variable. Comparison of terms in nonadjacent groups is unnecessary because such terms will always differ in at least two variables and cannot be combined using XY + XY′ = X. Similarly, the comparison of terms within a group is unnecessary because two terms with the same number of 1's must differ in at least two variables. Thus, only terms in adjacent groups must be compared. First, we will compare the term in group 0 with all of the terms in group 1. Terms 0000 and 0001 can be combined to eliminate the fourth variable, which yields 000–. Similarly, 0 and 2 combine to form 00–0 (a′b′d′), and 0 and 8 combine to form –000 (b′c′d′). The resulting terms are listed in Column II of Table 6-1. Whenever two terms combine, the corresponding decimal numbers differ by a power of 2 (1, 2, 4, 8, etc.). This is true because when the binary representations differ in exactly one column and if we subtract these binary representations, we get a 1 only in the column in which the difference exists. A binary number with a 1 in exactly one column is a power of 2. Column I TABLE 6-1 Determination of Prime Implicants group 0 © Cengage Learning 2014 group 1 Column II 0 1 2 8 0000 0001 0010 1000 ✓ ✓ ✓ ✓ group 2 5 6 9 10 0101 0110 1001 1010 ✓ ✓ ✓ ✓ group 3 7 0111 ✓ 14 1110 ✓ Column III 0, 1 000– ✓ 0, 2 00–0 ✓ 0, 8 –000 ✓ 1, 5 0–01 1, 9 2, 6 2, 10 8, 9 8, 10 5, 7 6, 7 6, 14 10, 14 –001 0–10 –010 100– 10–0 01–1 011– –110 1–10 ✓ ✓ ✓ ✓ ✓ 0, 1, 8, 9 0, 2, 8, 10 0, 8, 1, 9 0, 8, 2, 10 2, 6, 10, 14 2, 10, 6, 14 –00– –0–0 –00– –0–0 - - 10 - - 10 ✓ ✓ Because the comparison of group 0 with groups 2 and 3 is unnecessary, we proceed to compare terms in groups 1 and 2. Comparing term 1 with all terms in group 2, we find that it combines with 5 and 9 but not with 6 or 10. Similarly, term 2 combines only with 6 and 10, and term 8 only with 9 and 10. The resulting terms are listed in Column II. Each time a term is combined with another term, it is checked off. A term may be used more than once because X + X = X. Even though two terms have already been combined with other terms, they still must be compared and combined if possible. This is necessary because the resultant term may be needed to form the Quine-McCluskey Method 175 minimum sum solution. At this stage, we may generate redundant terms, but these redundant terms will be eliminated later. We finish with Column I by comparing terms in groups 2 and 3. New terms are formed by combining terms 5 and 7, 6 and 7, 6 and 14, and 10 and 14. Note that the terms in Column II have been divided into groups, according to the number of 1's in each term. Again, we apply XY + XY′ = X to combine pairs of terms in Column II. In order to combine two terms, the terms must have the same variables, and the terms must differ in exactly one of these variables. Thus, it is necessary only to compare terms which have dashes (missing variables) in corresponding places and which differ by exactly one in the number of 1's. Terms in the first group in Column II need only be compared with terms in the second group which have dashes in the same places. Term 000– (0, 1) combines only with term 100– (8, 9) to yield –00–. This is algebraically equivalent to a′b′c + ab′c′ = b′c′. The resulting term is listed in Column III along with the designation 0, 1, 8, 9 to indicate that it was formed by combining minterms 0, 1, 8, and 9. Term (0, 2) combines only with (8, 10), and term (0, 8) combines with both (1, 9) and (2, 10). Again, the terms which have been combined are checked off. Comparing terms from the second and third groups in Column II, we find that (2, 6) combines with (10, 14), and (2, 10) combines with (6, 14). Note that there are three pairs of duplicate terms in Column III. These duplicate terms were formed in each case by combining the same set of four minterms in a different order. After deleting the duplicate terms, we compare terms from the two groups in Column III. Because no further combination is possible, the process terminates. In general, we would keep comparing terms and forming new groups of terms and new columns until no more terms could be combined. The terms which have not been checked off because they cannot be combined with other terms are called prime implicants. Because every minterm has been included in at least one of the prime implicants, the function is equal to the sum of its prime implicants. In this example we have f = a′c′d + a′bd + a′bc + (1, 5) (5, 7) (6, 7) b′c′ + (0, 1, 8, 9) b′d′ + (0, 2, 8, 10) cd′ (2, 6, 10, 14) (6-3) In this expression, each term has a minimum number of literals, but the number of terms is not minimum. Using the consensus theorem to eliminate redundant terms yields f = a′bd + b′c′ + cd′ (6-4) which is the minimum sum-of-products expression for f . Section 6.2 discusses a better method of eliminating redundant prime implicants using a prime implicant chart. Next, we will define implicant and prime implicant and relate these terms to the Quine-McCluskey method. 176 Unit 6 Definition Given a function F of n variables, a product term P is an implicant of F iff for every combination of values of the n variables for which P = 1, F is also equal to 1. In other words, if for some combination of values of the variables, P = 1 and F = 0, then P is not an implicant of F. For example, consider the function F(a, b, c) = a′b′c′ + ab′c′ + ab′c + abc = b′c′ + ac (6-5) If a′b′c′ = 1, then F = 1; if ac = 1, then F = 1; etc. Hence, the terms a′b′c′, ac, etc., are implicants of F. In this example, bc is not an implicant of F because when a = 0 and b = c = 1, bc = 1 and F = 0. In general, if F is written in sum-of-products form, every product term is an implicant. Every minterm of F is also an implicant of F, and so is any term formed by combining two or more minterms. For example, in Table 6-1, all of the terms listed in any of the columns are implicants of the function given in Equation (6-2). Definition A prime implicant of a function F is a product term implicant which is no longer an implicant if any literal is deleted from it. In Equation (6-5), the implicant a′b′c′ is not a prime implicant because a′ can be eliminated, and the resulting term (b′c′) is still an implicant of F. The implicants b′c′ and ac are prime implicants because if we delete a literal from either term, the term will no longer be an implicant of F. Each prime implicant of a function has a minimum number of literals in the sense that no more literals can be eliminated from it by combining it with other terms. The Quine-McCluskey method, as previously illustrated, finds all of the product term implicants of a function. The implicants which are nonprime are checked off in the process of combining terms so that the remaining terms are prime implicants. A minimum sum-of-products expression for a function consists of a sum of some (but not necessarily all) of the prime implicants of that function. In other words, a sum-of-products expression which contains a term which is not a prime implicant cannot be minimum. This is true because the nonprime term does not contain a minimum number of literals—it can be combined with additional minterms to form a prime implicant which has fewer literals than the nonprime term. Any nonprime term in a sum-of-products expression can thus be replaced with a prime implicant, which reduces the number of literals and simplifies the expression. 6.2 The Prime Implicant Chart Given all the prime implicants of a function, the prime implicant chart can be used to select a minimum set of prime implicants. The minterms of the function are listed across the top of the chart, and the prime implicants are listed down the side. Quine-McCluskey Method 177 A prime implicant is equal to a sum of minterms, and the prime implicant is said to cover these minterms. If a prime implicant covers a given minterm, an X is placed at the intersection of the corresponding row and column. Table 6-2 shows the prime implicant chart derived from Table 6-1. All of the prime implicants (terms which have not been checked off in Table 6-1) are listed on the left. In the first row, X's are placed in columns 0, 1, 8, and 9, because prime implicant b′c′ was formed from the sum of minterms 0, 1, 8, and 9. Similarly, X's are placed in columns 0, 2, 8, and 10 opposite the prime implicant b′d′ and so forth. TABLE 6-2 Prime Implicant Chart © Cengage Learning 2014 (0, 1, 8, 9) (0, 2, 8, 10) (2, 6, 10, 14) (1, 5) (5, 7) (6, 7) b′c′ b′d′ cd′ a′c′d a′bd a′bc 0 1 2 5 6 7 8 9 10 14 ×× × ⊗ × × × × × × × ⊗ × × × × × × If a minterm is covered by only one prime implicant, then that prime implicant is called an essential prime implicant and must be included in the minimum sum of products. Essential prime implicants are easy to find using the prime implicant chart. If a given column contains only one X, then the corresponding row is an essential prime implicant. In Table 6-2, columns 9 and 14 each contain one X, so prime implicants b′c′ and cd′ are essential. Each time a prime implicant is selected for inclusion in the minimum sum, the corresponding row should be crossed out. After doing this, the columns which correspond to all minterms covered by that prime implicant should also be crossed out. Table 6-3 shows the resulting chart when the essential prime implicants and the corresponding rows and columns of Table 6-2 are crossed out. A minimum set of prime implicants must now be chosen to cover the remaining columns. In this example, a′bd covers the remaining two columns, so it is chosen. The resulting minimum sum of products is f = b′c′ + cd′ + a′bd which is the same as Equation (6-4). Note that even though the term a′bd is included in the minimum sum of products, a′bd is not an essential prime implicant. It is the sum of minterms m5 and m7; m5 is also covered by a′c′d, and m7 is also covered by a′bc. TABLE 6-3 © Cengage Learning 2014 (0, 1, 8, 9) (0, 2, 8, 10) (2, 6, 10, 14) (1, 5) (5, 7) (6, 7) b′c′ b′d′ cd′ a′c′d a′bd a′bc 0 1 2 5 6 7 8 9 10 × × × × × × × × × × × × × × × × × 14 × 178 Unit 6 When selecting prime implicants for a minimum sum, the essential prime implicants are chosen first because all essential prime implicants must be included in every minimum sum. After the essential prime implicants have been chosen, the minterms which they cover can be eliminated from the prime implicant chart by crossing out the corresponding columns. If the essential prime implicants do not cover all of the minterms, then additional nonessential prime implicants are needed. In simple cases, the nonessential prime implicants needed to form the minimum solution may be selected by trial and error. For larger prime implicant charts, additional procedures for chart reduction can be employed.1 (Also, see Problem 6.21.) Some functions have two or more minimum sum-of-products expressions, each having the same number of terms and literals. The next example shows such a function. Example A prime implicant chart which has two or more X's in every column is called a cyclic prime implicant chart. The following function has such a chart: F = Σ m(0, 1, 2, 5, 6, 7) (6-6) Derivation of prime implicants: 0 1 2 5 6 7 000 001 010 101 110 111 ✓ ✓ ✓ ✓ ✓ ✓ 0, 1 0, 2 1, 5 2, 6 5, 7 6, 7 00– 0–0 –01 –10 1–1 11– Table 6-4 shows the resulting prime implicant chart. All columns have two X's, so we will proceed by trial and error. Both (0, 1) and (0, 2) cover column 0, so we will try (0, 1). After crossing out row (0, 1) and columns 0 and 1, we examine column 2, which is covered by (0, 2) and (2, 6). The best choice is (2, 6) because it covers two of the remaining columns while (0, 2) covers only one of the remaining columns. After crossing out row (2, 6) and columns 2 and 6, we see that (5, 7) covers the remaining columns and completes the solution. Therefore, one solution is F = a′b′ + bc′ + ac. TABLE 6-4 ① → (0, 1) a′b′ © Cengage Learning 2014 ② ③ 1 (0, 2) (1, 5) → (2, 6) → (5, 7) (6, 7) a′c′ b′c bc′ ac ab 0 1 2 5 6 7 × × × × × × × × × × ×× For a discussion of such procedures, see E. J. McCluskey, Logic Design Principles (Prentice-Hall, 1986). Quine-McCluskey Method 179 However, we are not guaranteed that this solution is minimum. We must go back and solve the problem over again starting with the other prime implicant that covers column 0. The resulting table (Table 6-5) is TABLE 6-5 © Cengage Learning 2014 P1 P2 P3 P4 P5 P6 (0, 1) (0, 2) (1, 5) (2 6) (5, 7) (6, 7) a′b′ a′c′ b′c bc′ ac ab 0 1 2 5 6 7 × × × × × × × × × × × × Finish the solution and show that F = a′c′ + b′c + ab. Because this has the same number of terms and same number of literals as the expression for F derived in Table 6-4, there are two minimum sum-of-products solutions to this problem. Compare these two minimum solutions for Equation (6-6) with the solutions obtained in Figure 5-9 using Karnaugh maps. Note that each minterm on the map can be covered by two different loops. Similarly, each column of the prime implicant chart (Table 6-4) has two X's, indicating that each minterm can be covered by two different prime implicants. 6.3 Petrick's Method Petrick's method is a technique for determining all minimum sum-of-products solutions from a prime implicant chart. The example shown in Tables 6-4 and 6-5 has two minimum solutions. As the number of variables increases, the number of prime implicants and the complexity of the prime implicant chart may increase significantly. In such cases, a large amount of trial and error may be required to find the minimum solution(s). Petrick's method is a more systematic way of finding all minimum solutions from a prime implicant chart than the method used previously. Before applying Petrick's method, all essential prime implicants and the minterms they cover should be removed from the chart. We will illustrate Petrick's method using Table 6-5. First, we will label the rows of the table P1, P2, P3, etc. We will form a logic function, P, which is true when all of the minterms in the chart have been covered. Let P1 be a logic variable which is true when the prime implicant in row P1 is included in the solution, P2 be a logic variable which is true when the prime implicant in row P2 is included in the solution, etc. Because column 0 has X's in rows P1 and P2, we must choose row P1 or P2 in order to cover minterm 0. Therefore, the expression (P1 + P2) must be true. In order to cover minterm 1, we must choose row P1 or P3; therefore, (P1 + P3) must be true. In order 180 Unit 6 to cover minterm 2, (P2 + P4) must be true. Similarly, in order to cover minterms 5, 6, and 7, the expressions (P3 + P5), (P4 + P6) and (P5 + P6) must be true. Because we must cover all of the minterms, the following function must be true: P = (P1 + P2)(P1 + P3)(P2 + P4)(P3 + P5)(P4 + P6)(P5 + P6) = 1 The expression for P in effect means that we must choose row P1 or P2, and row P1 or P3, and row P2 or P4, etc. The next step is to reduce P to a minimum sum of products. This is easy because there are no complements. First, we multiply out, using (X + Y)(X + Z) = X + YZ and the ordinary distributive law: P = (P1 + P2P3)(P4 + P2P6)(P5 + P3P6) = (P1P4 + P1P2P6 + P2P3P4 + P2P3P6)(P5 + P3P6) = P1P4P5 + P1P2P5P6 + P2P3P4P5 + P2P3P5P6 + P1P3P4P6 + P1P2P3P6 + P2P3P4P6 + P2P3P6 Next, we use X + XY = X to eliminate redundant terms from P, which yields P = P1P4P5 + P1P2P5P6 + P2P3P4P5 + P1P3P4P6 + P2P3P6 Because P must be true (P = 1) in order to cover all of the minterms, we can translate the equation back into words as follows. In order to cover all of the minterms, we must choose rows P1 and P4 and P5, or rows P1 and P2 and P5 and P6, or . . . or rows P2 and P3 and P6. Although there are five possible solutions, only two of these have the minimum number of rows. Thus, the two solutions with the minimum number of prime implicants are obtained by choosing rows P1, P4, and P5 or rows P2, P3, and P6. The first choice leads to F = a′b′ + bc′ + ac, and the second choice to F = a′c′ + b′c + ab, which are the two minimum solutions derived in Section 6.2. In summary, Petrick's method is as follows: 1. 2. 3. 4. 5. 6. Reduce the prime implicant chart by eliminating the essential prime implicant rows and the corresponding columns. Label the rows of the reduced prime implicant chart P1, P2, P3, etc. Form a logic function P which is true when all columns are covered. P consists of a product of sum terms, each sum term having the form (Pi0 + Pi1 + · · ·), where Pi0, Pi1 . . . represent the rows which cover column i. Reduce P to a minimum sum of products by multiplying out and applying X + XY = X. Each term in the result represents a solution, that is, a set of rows which covers all of the minterms in the table. To determine the minimum solutions (as defined in Section 5.1), find those terms which contain a minimum number of variables. Each of these terms represents a solution with a minimum number of prime implicants. For each of the terms found in step 5, count the number of literals in each prime implicant and find the total number of literals. Choose the term or terms which correspond to the minimum total number of literals, and write out the corresponding sums of prime implicants. Quine-McCluskey Method 181 The application of Petrick's method is very tedious for large charts, but it is easy to implement on a computer. 6.4 Simplification of Incompletely Specified Functions Given an incompletely specified function, the proper assignment of values to the don't-care terms is necessary in order to obtain a minimum form for the function. In this section, we will show how to modify the Quine-McCluskey method in order to obtain a minimum solution when don't-care terms are present. In the process of finding the prime implicants, we will treat the don't-care terms as if they were required minterms. In this way, they can be combined with other minterms to eliminate as many literals as possible. If extra prime implicants are generated because of the don't-cares, this is correct because the extra prime implicants will be eliminated in the next step anyway. When forming the prime implicant chart, the don'tcares are not listed at the top. This way, when the prime implicant chart is solved, all of the required minterms will be covered by one of the selected prime implicants. However, the don't-care terms are not included in the final solution unless they have been used in the process of forming one of the selected prime implicants. The following example of simplifying an incompletely specified function should clarify the procedure. F(A, B, C, D) = Σ m(2, 3, 7, 9, 11, 13) + Σ d(1, 10, 15) (the terms following d are don't-care terms) The don't-care terms are treated like required minterms when finding the prime implicants: 1 2 3 9 10 7 11 13 5 0001 0010 0011 1001 1010 0111 1011 1101 1111 ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ (1, 3) 00–1 ✓ (1, 9) –001 ✓ (2, 3) 001– ✓ (2, 10) –010 ✓ (3, 7) 0–11 ✓ (3, 11) –011 ✓ (9, 11) 10–1 ✓ (9, 13) 1–01 ✓ (10, 11) 101– ✓ (7, 15) –111 ✓ (11, 15) 1–11 ✓ (13, 15) 11–1 ✓ (1, 3, 9, 11) –0–1 (2, 3, 10,11) –01– (3, 7, 11, 15) - - 11 (9, 11, 13, 15) 1 - - 1 182 Unit 6 The don't-care columns are omitted when forming the prime implicant chart: 2 3 (1, 3, 9, 11) *(2, 3, 10, 11) *(3, 7, 11, 15) *(9, 11, 13, 15) 7 9 11 13 × × × × × × × × × × × × F = B′C + CD + AD × *Indicates an essential prime implicant. Note that although the original function was incompletely specified, the final simplified expression for F is defined for all combinations of values for A, B, C, and D and is therefore completely specified. In the process of simplification, we have automatically assigned values to the don't-cares in the original truth table for F. If we replace each term in the final expression for F by its corresponding sum of minterms, the result is F = (m2 + m3 + m10 + m11) + (m3 + m7 + m11 + m15) + (m9 + m11 + m13 + m15) Because m10 and m15 appear in this expression and m1 does not, this implies that the don't-care terms in the original truth table for F have been assigned as follows: for ABCD = 0001, F = 0; for 1010, F = 1; for 1111, F = 1 6.5 Simplification Using Map-Entered Variables Although the Quine-McCluskey method can be used with functions with a fairly large number of variables, it is not very efficient for functions that have many variables and relatively few terms. Some of these functions can be simplified by using a modification of the Karnaugh map method. By using map-entered variables, Karnaugh map techniques can be extended to simplify functions with more than four or five variables. Figure 6-1(a) shows a four-variable map with two additional variables entered in the squares in the map. When E appears in a square, this means FIGURE 6-1 Use of MapEntered Variables © Cengage Learning 2014 AB CD AB 00 01 11 00 1 01 X E X 11 1 E 1 10 1 10 CD AB 00 01 11 00 1 F 01 X X 1 11 1 1 X 10 1 10 CD AB 00 01 11 00 X 01 X 1 X 1 11 X 1 X X 10 X 10 CD 00 01 11 10 00 X 01 X X 1 X 11 X X X X 10 X X G E=F=0 MS0 = A′B′ + ACD E = 1, F = 0 MS1 = A′D E = 0, F = 1 MS2 = AD (a) (b) (c) (d) Quine-McCluskey Method 183 that if E = 1, the corresponding minterm is present in the function G, and if E = 0, the minterm is absent. Thus, the map represents the six-variable function G(A, B, C, D, E, F) = m0 + m2 + m3 + Em5 + Em7 + Fm9 + m11 + m15 (+ don't-care terms) where the minterms are minterms of the variables A, B, C, and D. Note that m9 is present in G only when F = 1. We will now use a three-variable map to simplify the function: F(A, B, C, D) = A′B′C + A′BC + A′B′C′D + ABCD + (AB′C) where the AB′C is a don't-care term. Because D appears in only two terms, we will choose it as a map-entered variable, which leads to Figure 6-2(a). We will simplify F by first considering D = 0 and then D = 1. First set D = 0 on the map, and F reduces to A′C. Setting D = 1 leads to the map of Figure 6-2(b). The two 1's on the original map have already been covered by the term A′C, so they are changed to X's because we do not care whether they are covered again or not. From Figure 6-2(b), when D = 1. Thus, the expression F = A′C + D(C + A′B) = A′C + CD + A′BD gives the correct value of F both when D = 0 and when D = 1. This is a minimum expression for F, as can be verified by plotting the original function on a four-variable map; see Figure 6-2(c). Next, we will discuss a general method of simplifying functions using map-entered variables. In general, if a variable Pi is placed in square mj of a map of function F, this means that F = 1 when Pi = 1, and the variables are chosen so that mj = 1. Given a map with variables P1, P2, . . . entered into some of the squares, the minimum sum-ofproducts form of F can be found as follows: Find a sum-of-products expression for F of the form F = MS0 + P1MS1 + P2MS2 + · · · where MS0 is the minimum sum obtained by setting P1 = P2 = · · · = 0. FIGURE 6-2 Simplification Using a Map-Entered Variable © Cengage Learning 2014 A BC A 0 1 00 BC DA 0 1 00 BC 00 01 11 10 X X 1 1 1 00 01 1 X 01 X X 01 1 11 1 D 11 X 1 11 1 10 D 10 1 (a) 1 10 (b) (c) 184 Unit 6 MS1 is the minimum sum obtained by setting P1 = 1, Pj = 0 (j ≠ 1), and replacing all 1's on the map with don't-cares. MS2 is the minimum sum obtained by setting P2 = 1, Pj = 0 (j ≠ 2) and replacing all 1's on the map with don't-cares. (Corresponding minimum sums can be found in a similar way for any remaining map-entered variables.) The resulting expression for F will always be a correct representation of F. This expression will be minimum provided that the values of the map-entered variables can be assigned independently. On the other hand, the expression will not generally be minimum if the variables are not independent (for example, if P1 = P2′). For the example of Figure 6-1(a), maps for finding MS0, MS1, and MS2 are shown in Figures 6-1(b), (c), and (d), where E corresponds to P1 and F corresponds to P2. The resulting expression is a minimum sum of products for G: G = A′B′ + ACD + EA′D + FAD After some practice, it should be possible to write the minimum expression directly from the original map without first plotting individual maps for each of the minimum sums. 6.6 Conclusion We have discussed four methods for reducing a switching expression to a minimum sumof-products or a minimum product-of-sums form: algebraic simplification, Karnaugh maps, Quine-McCluskey method, and Petrick's method. Many other methods of simplification are discussed in the literature, but most of these methods are based on variations or extensions of the Karnaugh map or Quine-McCluskey techniques. Karnaugh maps are most useful for functions with three to five variables. The Quine-McCluskey technique can be used with a high-speed digital computer to simplify functions with up to 15 or more variables. Such computer programs are of greatest value when used as part of a computer-aided design (CAD) package that assists with deriving the equations as well as implementing them. Algebraic simplification is still valuable in many cases, especially when different forms of the expressions are required. For problems with a large number of variables and a small number of terms, it may be impossible to use the Karnaugh map, and the Quine-McCluskey method may be very cumbersome. In such cases, algebraic simplification may be the easiest method to use. In situations where a minimum solution is not required or where obtaining a minimum solution requires too much computation to be practical, heuristic procedures may be used to simplify switching functions. One of the more popular heuristic procedures is the Espresso-II method,2 which can produce near minimum solutions for a large class of problems. The minimum sum-of-products and minimum product-of-sums expressions we have derived lead directly to two-level circuits that use a minimum number of AND 2 This method is described in R. K. Brayton et al., Logic Minimization Algorithms for VLSI Synthesis (Kluwer Academic Publishers, 1984). Quine-McCluskey Method 185 and OR gates and have a minimum number of gate inputs. As discussed in Unit 7, these circuits are easily transformed into circuits that contain NAND or NOR gates. These minimum expressions may also be useful when designing with some types of array logic, as discussed in Unit 9. However, many situations exist where minimum expressions do not lead to the best design. For practical designs, many other factors must be considered, such as the following: What is the maximum number of inputs a gate can have? What is the maximum number of outputs a gate can drive? Is the speed with which signals propagate through the circuit fast enough? How can the number of interconnections in the circuit be reduced? Does the design lead to a satisfactory circuit layout on a printed circuit board or on a silicon chip? Until now, we have considered realizing only one switching function at a time. Unit 7 describes design techniques and Unit 9 describes components that can be used when several functions must be realized by a single circuit. Programmed Exercise 6.1 Cover the answers to this exercise with a sheet of paper and slide it down as you check your answers. Find a minimum sum-of-products expression for the following function: f(A, B, C, D, E) = Σ m(0, 2, 3, 5, 7, 9, 11, 13, 14, 16, 18, 24, 26, 28, 30) Translate each decimal minterm into binary and sort the binary terms into groups according to the number of 1's in each term. Answer: 0 2 16 3 5 9 18 24 7 11 13 14 26 28 30 00000 ✓ 0,2 00010 ✓ 10000 00011 00101 01001 10010 11000 00111 01011 01101 01110 11010 11100 11110 000-0 Compare pairs of terms in adjacent groups and combine terms where possible. (Check off terms which have been combined.) 186 Unit 6 Answer: 0 2 16 3 5 9 18 24 7 11 13 14 26 28 30 00000 00010 10000 00011 00101 01001 10010 11000 00111 01101 01101 01110 11010 11100 11110 0, 2 ✓ 0, 16 ✓ 2, 3 ✓ 2, 18 ✓ ✓ 16, 18 ✓ 16, 24 3, 7 ✓ 3, 11 ✓ 5, 7 ✓ ✓ 5, 13 ✓ 9, 11 ✓ 9, 13 ✓ ✓ 18, 26 ✓ 24, 26 ✓ 24, 28 14, 30 26, 30 28, 30 000−0 ✓ −0000 0001− −0010 100−0 ✓ 1−000 00−11 0−011 001−1 0−101 010−1 01−01 1−010 110−0 11−00 −1110 11−10 0, 2, 16, 18 −00−0 Now, compare pairs of terms in adjacent groups in the second column and combine terms where possible. (Check off terms which have been combined.) Check your work by noting that each new term can be formed in two ways. (Cross out duplicate terms.) Answer: (third column) 0, 2, 16, 18 –00–0 16, 18, 24, 26 1–0–0 24, 26, 28, 30 11 -- 0 (check off (0, 2), (16, 18), (0, 16), and (2, 18)) (check off (16, 18), (24, 26), (16, 24), and (18, 26)) (check off (24, 26), (28, 30), (24, 28), and (26, 30)) Can any pair of terms in the third column be combined? Complete the given prime implicant chart. 0 (0, 2, 16, 18) 2 Quine-McCluskey Method Answer: 187 No pair of terms in the third column combine. 0 2 (0, 2, 16, 18) (16, 18, 24, 26) (24, 26, 28, 30) (2, 3) (3, 7) (3, 11) (5, 7) (5, 13) (9, 11) (9, 13) (14, 30) × × 3 × × × 5 7 9 11 13 14 16 18 24 26 28 30 × Determine the essential prime implicants, and cross out the corresponding rows and columns. 0 2 3 5 7 9 11 13 14 16 18 24 26 28 30 Answer: *(0, 2, 16, 18) (16, 18, 24, 26) *(24, 26, 28, 30) (2, 3) (3, 7) (3, 11) (5, 7) (5, 13) (9, 11) (9, 13) *(14, 30) ×× ×× × × × × ×× ×× × × × × × *Indicates an essential prime implicant. Note that all remaining columns contain two or more X's. Choose the first column which has two X's and then select the prime implicant which covers the first X in that column. Then, choose a minimum number of prime implicants which cover the remaining columns in the chart. 188 Unit 6 Answer: 0 2 3 5 7 9 11 13 14 16 18 24 26 28 30 × × × × × × × × × × *(0, 2, 16, 18) × × (16, 18, 24, 26) *(24, 26, 28, 30) (2, 3) (3, 7) → (3, 11) → (5, 7) (5, 13) (9, 11) → (9, 13) *(14, 30) ×× × × × ×× × × × ×× × × × × *Indicates an essential prime implicant. From this chart, write down the chosen prime implicants in 0, 1, and – notation. Then, write the minimum sum of products in algebraic form. Answer: –00–0, 11--0, 0–011, 001–1, 01–01, and –1110 f = B′C′E′ + ABE′ + A′C′DE + A′B′CE + A′BD′E + BCDE′ The prime implicant chart with the essential prime implicants crossed out is repeated here. Find a second minimum sum-of-products solution. 0 2 3 5 7 9 11 13 14 16 18 24 26 28 30 *(0, 2, 16, 18) (16, 18, 24, 26) *(24, 26, 28, 30) (2, 3) (3, 7) (3, 11) (5, 7) (5, 13) (9, 11) (9, 13) *(14, 30) ×× ×× × × ×× × × × × × × × × × × × × × × × × × *Indicates an essential prime implicant. Answer: Start by choosing prime implicant (5, 13). f = BCDE′ + B′C′E′ + ABE′ + A′B′DE + A′CD′E + A′BC′E Quine-McCluskey Method 189 Problems 6.2 For each of the following functions, find all of the prime implicants, using the QuineMcCluskey method. (a) f(a, b, c, d) = Σ m(1, 5, 7, 9, 11, 12, 14, 15) (b) f(a, b, c, d) = Σ m(0, 1, 3, 5, 6, 7, 8, 10, 14, 15) 6.3 Using a prime implicant chart, find all minimum sum-of-products solutions for each of the functions given in Problem 6.2. 6.4 For this function, find a minimum sum-of-products solution, using the QuineMcCluskey method. f(a, b, c, d) = Σ m(1, 3, 4, 5, 6, 7, 10, 12, 13) + Σ d(2, 9, 15) 6.5 Find all prime implicants of the following function and then find all minimum solutions using Petrick's method: F(A, B, C, D) = Σ m(9, 12, 13, 15) + Σ d(1, 4, 5, 7, 8, 11, 14) 6.6 Using the method of map-entered variables, use four-variable maps to find a minimum sum-of-products expression for (a) F(A, B, C, D, E) = Σ m(0, 4, 5, 7, 9) + Σ d(6, 11) + E(m1 + m15), where the m's represent minterms of the variables A, B, C, and D. (b) Z(A, B, C, D, E, F, G) = Σ m(0, 3, 13, 15) + Σ d(1, 2, 7, 9, 14) + E(m6 + m8) + Fm12 + Gm5 6.7 For each of the following functions, find all of the prime implicants using the QuineMcCluskey method. (a) f(a, b, c, d) = Σ m(0, 3, 4, 5, 7, 9, 11, 13) (b) f(a, b, c, d) = Σ m(2, 4, 5, 6, 9, 10, 11, 12, 13, 15) 6.8 Using a prime implicant chart, find all minimum sum-of-products solutions for each of the functions given in Problem 6.7. 6.9 For each function, find a minimum sum-of-products solution using the QuineMcCluskey method. (a) f(a, b, c, d) = Σ m(2, 3, 4, 7, 9, 11, 12, 13, 14) + Σ d(1, 10, 15) (b) f(a, b, c, d) = Σ m(0, 1, 5, 6, 8, 9, 11, 13) + Σ d(7, 10, 12) (c) f(a, b, c, d) = Σ m(3, 4, 6, 7, 8, 9, 11, 13, 14) + Σ d(2, 5, 15) 6.10 Work Problem 5.24(a) using the Quine-McCluskey method. 6.11 F(A, B, C, D, E) = Σ m(0, 2, 6, 7, 8, 10, 11, 12, 13, 14, 16, 18, 19, 29, 30) + Σ d(4, 9, 21) 190 Unit 6 Find the minimum sum-of-products expression for F, using the Quine-McCluskey method. Underline the essential prime implicants in this expression. 6.12 Using the Quine-McCluskey method, find all minimum sum-of-products expressions for (a) f(A, B, C, D, E) = Σ m(0, 1, 2, 3, 4, 8, 10, 11, 19, 21, 22, 23, 27, 28, 29, 30) (b) f(A, B, C, D, E) = Σ m(0, 1, 2, 4, 8, 11, 13, 14, 15, 17, 18, 20, 21, 26, 27, 30, 31) 6.13 Using the Quine-McCluskey method, find all minimum product-of-sums expressions for the functions of Problem 6.12. 6.14 (a) Using the Quine-McCluskey, method find all prime implicants of f(A, B, C, D) = Σ m(1, 3, 5, 6, 8, 9, 12, 14, 15) + Σ d(4, 10, 13). Identify all essential prime implicants and find all minimum sum-of-products expressions. (b) Repeat part (a) for f ′. 6.15 (a) Use the Quine-McCluskey method to find all prime implicants of f(a, b, c, d, e) = Σ m(1, 2, 4, 5, 6, 7, 9, 12, 13, 15, 17, 20, 22, 25, 28, 30). Find all essential prime implicants, and find all minimum sum-of-products expressions. (b) Repeat part (a) for f ′. 6.16 G(A, B, C, D, E, F) = Σ m(1, 2, 3, 16, 17, 18, 19, 26, 32, 39, 48, 63) + Σ d(15, 28, 29, 30) (a) Find all minimum sum-of-products expressions for G. (b) Circle the essential prime implicants in your answer. (c) If there were no don't-care terms present in the original function, how would your answer to part (a) change? (Do this by inspection of the prime implicant chart; do not rework the problem.) 6.17 (a) Use the Quine-McCluskey procedure to find all prime implicants of the function G(A, B, C, D, E, F) = Σ m(1, 7, 11, 12, 15, 33, 35, 43, 47, 59, 60) + Σ d(30, 50, 54, 58). Identify all essential prime implicants and find all minimum sum-of-products expressions. (b) Repeat part (a) for G′. 6.18 The following prime implicant table (chart) is for a four-variable function f(A, B, C, D). (a) Give the decimal representation for each of the prime implicants. (b) List the maxterms of f . (c) List the don't-cares of f , if any. (d) Give the algebraic expression for each of the essential prime implicants. 2 –0–1 –01– - - 11 1--1 × 3 × × × 7 × 9 × × 11 × × × × 13 × Quine-McCluskey Method 191 6.19 Packages arrive at the stockroom and are delivered on carts to offices and laboratories by student employees. The carts and packages are various sizes and shapes. The students are paid according to the carts used. There are five carts and the pay for their use is Cart C1: $2 Cart C2: $1 Cart C3: $4 Cart C4: $2 Cart C5: $2 On a particular day, seven packages arrive, and they can be delivered using the five carts as follows: C1 can be used for packages P1, P3, and P4. C2 can be used for packages P2, P5, and P6. C3 can be used for packages P1, P2, P5, P6, and P7. C4 can be used for packages P3, P6, and P7. C5 can be used for packages P2 and P4. The stockroom manager wants the packages delivered at minimum cost. Using minimization techniques described in this unit, present a systematic procedure for finding the minimum cost solution. 6.20 Use the Quine-McCluskey procedure to find all prime implicants of the function h(A, B, C, D, E, F, G) = Σ m(24, 28, 39, 47, 70, 86, 88, 92, 102, 105, 118). Express the prime implicants algebraically. 6.21 Shown below is the prime implicant chart for a completely specified four-variable combinational logic function r(w, x, y, z). (a) Algebraically express r as a product of maxterms. (b) Give algebraic expressions for the prime implicants labeled A, C, and D in the table. (c) Find all minimal sum-of-product expressions for r. You do not have to give algebraic expressions; instead just list the prime implicants (A, B, C, etc.) required in the sum(s). 0 A B C D E F G H 4 × × × 5 6 7 × × × × × × × × × 8 9 10 11 13 × × × × × × × × × × × 14 15 × × × × × × 6.22 (a) In the prime implicant chart of Problem 6.21, column 7 is said to cover column 6 since column 7 has an X in each row that column 6 does. Similarly, column 11 192 Unit 6 covers column 10 and column 15 covers column 14. Columns 7, 11, and 15 can be removed to obtain a simpler chart having the same solutions as the original. Explain why this is correct. (b) In Table 6-5 (after removing row P2 and columns 0 and 2), row P3 covers row P1. Row (prime implicant) P1 can be removed, and the resulting chart will have a minimum solution for the original table. Explain why this is correct. Are there any restrictions on the two prime implicants to allow removal of the covered prime implicant? (c) After deleting row P1 from Table 6-5, row P3 must be included in a minimal solution for the chart. Why? 6.23 Find all prime implicants of the following function, and then find all minimum solutions using Petrick's method: F(A, B, C, D) = Σ m(7, 12, 14, 15) + Σ d(1, 3, 5, 8, 10, 11, 13) 6.24 Using the method of map-entered variables, use four-variable maps to find a minimum sum-of-products expression for (a) F(A, B, C, D, E) = Σ m(0, 4, 6, 13, 14) + Σ d(2, 9) + E(m1 + m12) (b) Z(A, B, C, D, E, F, G) = Σ m(2, 5, 6, 9) + Σ d(1, 3, 4, 13, 14) + E(m11 + m12) + F(m10) + G(m0) 6.25 (a) Rework Problem 6.6(a), using a five-variable map. (b) Rework Problem 6.6(a), using the Quine-McCluskey method. Note that you must express F in terms of minterms of all five variables; the original fourvariable minterms cannot be used. 6.26 Using map-entered variables, find the minimum sum-of-products expressions for the following function: G = C′E′F + DEF + AD′E′F ′ + BDE′F + AD′EF ′ UNIT Multi-Level Gate Circuits NAND and NOR Gates 7 Objectives 1. Design a minimal two-level or multi-level circuit of AND and OR gates to realize a given function. (Consider both circuits with an OR gate at the output and circuits with an AND gate at the output.) 2. Design or analyze a two-level gate circuit using any one of the eight basic forms (AND-OR, NAND-NAND, OR-NAND, NOR-OR, OR-AND, NOR-NOR, AND-NOR, and NAND-AND). 3. Design or analyze a multi-level NAND-gate or NOR-gate circuit. 4. Convert circuits of AND and OR gates to circuits of NAND gates or NOR gates, and conversely, by adding or deleting inversion bubbles. 5. Design a minimal two-level, multiple-output AND-OR, OR-AND, NANDNAND, or NOR-NOR circuit using Karnaugh maps. 193 194 Unit 7 Study Guide 1. Study Section 7.1, Multi-Level Gate Circuits. (a) What are two ways of changing the number of levels in a gate circuit? (b) By constructing a tree diagram, determine the number of gates, gate inputs, and levels of gates required to realize Z1 and Z2: Z1 = [(A + B)C + DE(F + G)] H Z2 = A + B [C + DE(F + G)] Check your answers by drawing the corresponding gate circuits. (c) In order to find a minimum two-level solution, why is it necessary to consider both a sum-of-products form and a product-of-sums form for the function? (d) One realization of Z = ABC(D + E) + FG is A B C Z D E F G Redraw the circuit so that it uses one less gate and so that the output of an AND gate never goes directly to the input of another AND gate. Multi-Level Gate Circuits NAND and NOR Gates 195 (e) Work Problems 7.1 and 7.2. Unless otherwise specified, you may always assume that both the variables and their complements are available as circuit inputs. 2. Study Section 7.2, NAND and NOR Gates. (a) For each gate, specify the missing inputs: 1 1 1 0 0 0 0 1 (b) What is meant by functionally complete set of logic gates? (c) How can you show that a set of logic gates is functionally complete? (d) Show that the NOR gate itself is functionally complete. (e) Using NAND gates, draw a circuit for F = (A′(BC)′)′. (f) Using NOR gates, draw a circuit for F = ((X + Y)′ + (X′ + Z)′)′. 3. Study Section 7.3, Design of Two-Level NAND- and NOR-Gate Circuits. (a) Draw the circuit corresponding to Equation (7-17). (b) Derive Equation (7-18). (c) Make sure that you understand the relation between Equations (7-13) through (7-21) and the diagrams of Figure 7-11. (d) Why is the NOR-NAND form degenerate? Unit 7 (e) What assumption is made about the types of inputs available when the procedures for designing two-level NAND-NAND and NOR-NOR circuits are used? (f) For these procedures the literal inputs to the output gate are complemented but not the literal inputs to the other gates. Explain why. Use an equation to illustrate. P1 y1 y2 P2 ℓ1 ℓ2 ... x1 x2 ... (g) A general OR-AND circuit follows. Transform this to a NOR-NOR circuit and prove that your transformation is valid. F ... ... 196 (h) Work Problem 7.3. 4. Study Section 7.4, Design of Multi-Level NAND- and NOR-Gate Circuits. (a) Verify that the NAND circuit of Figure 7-13 is correct by dividing the corresponding circuit of AND and OR gates into two-level subcircuits and transforming each subcircuit. (b) If you wish to design a two-level circuit using only NOR gates, should you start with a minimum sum of products or a minimum product of sums? (c) Note that direct conversion of a circuit of AND and OR gates to a NANDgate circuit requires starting with an OR gate at the output, but the direct conversion to a NOR-gate circuit requires starting with an AND gate at the output. This is easy to remember because a NAND is equivalent to an OR with the inputs inverted: a b c f = a′ b′ c′ f and a NOR is equivalent to an AND with the inputs inverted: a b c f = a′ b′ c′ f Multi-Level Gate Circuits NAND and NOR Gates 197 (d) Convert the circuit of Figure 7-1(b) to all NAND gates. (e) Work Problems 7.4, 7.5, 7.6, and 7.7. 5. Study Section 7.5, Circuit Conversion Using Alternative Gate Symbols. (a) Determine the logic function realized by each of the following circuits: A B A C F B C G G= F= (b) Convert the circuit of Figure 7-13(a) to NAND gates by adding bubbles and complementing input variables when necessary. (You should have added 12 bubbles. Your result should be similar to Figure 7-13(b), except some of the NAND gates will use the alternative symbol.) (c) Draw a circuit of AND and OR gates for the following equation: Z = A [BC + D + E(F + GH)] Then convert to NOR gates by adding bubbles and complementing inputs when necessary. (You should have added 10 bubbles and complemented six input variables.) (d) Work Problem 7.8. 6. Study Section 7.6, Design of Two-Level, Multiple-Output Circuits. (a) In which of the following cases would you replace a term xy′ with xy′z + xy′z′? (1) Neither xy′z or xy′z′ is used in another function. (2) Both xy′z and xy′z′ are used in other functions. (3) Term xy′z is used in another function, but xy′z′ is not. (b) In the second example (Figure 7-23), in f2, c could have been replaced by bc + b′c because bc and b′c were available "free" from f1 and f3. Why was this replacement not made? 198 Unit 7 (c) In the following example, compute the cost of realizing f1 and f2 separately; then compute the cost using the term a′b′c in common between the two functions. Use a two-level AND-OR circuit in both cases. a a 0 bc 1 00 01 1 11 1 0 1 00 1 1 01 1 bc 11 1 10 10 f1 1 1 f2 (d) Find expressions which correspond to a two-level, minimum multipleoutput, AND-OR realization of F1, F2, and F3. Why should the term cd not be included in F1? ab cd ab 00 01 11 10 cd ab 00 01 11 10 cd 00 1 00 1 00 01 1 01 1 01 1 11 11 1 1 1 1 10 1 1 1 1 1 10 F1 11 1 00 01 1 1 1 1 10 1 1 10 F2 F3 F1 = F2 = F3 = (e) Work Problems 7.9, 7.10, and 7.11. (f) Work Problem 7.12. (Hint: Work with the 0's on the maps and first find a minimum solution for f1′, f2′, and f3′.) 7. Study Section 7.7, Multiple-Output NAND- and NOR-Gate Circuits. (a) Derive expressions for the F1 and F2 outputs of the NOR circuits of Figure 7-26(b) by finding the equation for each gate output, and show that these expressions reduce to the original expressions for F1 and F2. Multi-Level Gate Circuits NAND and NOR Gates 199 (b) Convert Figure 7-26(a) to 7-26(b) by using the bubble method. (c) Work Problem 7.13. Multi-Level Gate Circuits NAND and NOR Gates In the first part of this unit, you will learn how to design circuits which have more than two levels of AND and OR gates. In the second part you will learn techniques for designing with NAND and NOR gates. These techniques generally consist of first designing a circuit of AND and OR gates and then converting it to the desired type of gates. These techniques are easy to apply provided that you start with the proper form of circuit. 7.1 Multi-Level Gate Circuits The maximum number of gates cascaded in series between a circuit input and the output is referred to as the number of levels of gates (not to be confused with voltage levels). Thus, a function written in sum-of-products form or in product-of-sums form corresponds directly to a two-level gate circuit. As is usually the case in digital circuits where the gates are driven from flip-flop outputs (as discussed in Unit 11), we will assume that all variables and their complements are available as circuit inputs. For this reason, we will not normally count inverters which are connected directly 200 Unit 7 to input variables when determining the number of levels in a circuit. In this unit we will use the following terminology: 1. 2. 3. 4. AND-OR circuit means a two-level circuit composed of a level of AND gates followed by an OR gate at the output. OR-AND circuit means a two-level circuit composed of a level of OR gates followed by an AND gate at the output. OR-AND-OR circuit means a three-level circuit composed of a level of OR gates followed by a level of AND gates followed by an OR gate at the output. Circuit of AND and OR gates implies no particular ordering of the gates; the output gate may be either AND or OR. The number of levels in an AND-OR circuit can usually be increased by factoring the sum-of-products expression from which it was derived. Similarly, the number of levels in an OR-AND circuit can usually be increased by multiplying out some of the terms in the product-of-sums expression from which it was derived. Logic designers are concerned with the number of levels in a circuit for several reasons. Sometimes factoring (or multiplying out) to increase the number of levels of gates will reduce the required number of gates and gate inputs and, thus, reduce the cost of building the circuit, but in other cases increasing the number of levels will increase the cost. In many applications, the number of gates which can be cascaded is limited by gate delays. When the input of a gate is switched, there is a finite time before the output changes. When several gates are cascaded, the time between an input change and the corresponding change in the circuit output may become excessive and slow down the operation of the digital system. The number of gates, gate inputs, and levels in a circuit can be determined by inspection of the corresponding expression. In the example of Figure 7-1(a), the tree diagram drawn below the expression for Z indicates that the corresponding circuit will have four levels, six gates, and 13 gate inputs, as verified in Figure 7-1(b). Each FIGURE 7-1 Four-Level Realization of Z Z = (AB + C) (D + E + FG) + H 2 2 A B F G Level 4 © Cengage Learning 2014 C 2 2 DE Level 3 3 Level 2 2 H 2 Level 1 Z (a) (b) Multi-Level Gate Circuits NAND and NOR Gates FIGURE 7-2 Three-Level Realization of Z Z = AB(D + E) + C(D + E) + ABFG + CFG + H 2 2 D E Level 3 * © Cengage Learning 2014 AB 3 201 2 4 3 C ABFG CFG Level 2 H 5 * Level 1 The same gate can be used for both appearances of (D + E). Z (a) (b) node on the tree diagram represents a gate, and the number of gate inputs is written beside each node. We can change the expression for Z to three levels by partially multiplying it out: Z = (AB + C) [(D + E) + FG] + H = AB(D + E) + C(D + E) + ABFG + CFG + H As shown in Figure 7-2, the resulting circuit requires three levels, six gates, and 19 gate inputs. Example of Multi-Level Design Using AND and OR Gates Solution: Problem: f(a, b, c, d) = Σ m(1, 5, 6, 10, 13, 14) Consider solutions with two levels of gates and three levels of gates. Try to minimize the number of gates and the total number of gate inputs. Assume that all variables and their complements are available as inputs. First, simplify f by using a Karnaugh map (Figure 7-3): FIGURE 7-3 © Cengage Learning 2014 Find a circuit of AND and OR gates to realize ab 00 01 11 10 00 0 0 0 0 01 1 1 1 0 11 0 0 0 0 10 0 1 1 1 cd f = a′c′d + bc′d + bcd′ + acd′ (7-1) 202 Unit 7 This leads directly to a two-level AND-OR gate circuit (Figure 7-4): FIGURE 7-4 © Cengage Learning 2014 a′ c′ d b c′ d b c d′ a c d′ f Two levels Five gates 16 gate inputs Factoring Equation (7-1) yields f = c′d(a′ + b) + cd′(a + b) (7-2) which leads to the following three-level OR-AND-OR gate circuit (Figure 7-5): FIGURE 7-5 a′ © Cengage Learning 2014 b a c′ d f c d′ Three levels Five gates 12 gate inputs b Both of these solutions have an OR gate at the output. A solution with an AND gate at the output might have fewer gates or gate inputs. A two-level OR-AND circuit corresponds to a product-of-sums expression for the function. This can be obtained from the 0's on the Karnaugh map as follows: f′ = c′d′ + ab′c′ + cd + a′b′c f = (c + d)(a′ + b + c)(c′ + d′)(a + b + c′) Equation (7-4) leads directly to a two-level OR-AND circuit (Figure 7-6): FIGURE 7-6 © Cengage Learning 2014 c d a′ b c c′ d′ a b c′ f Two levels Five gates 14 gate inputs (7-3) (7-4) Multi-Level Gate Circuits NAND and NOR Gates 203 To get a three-level circuit with an AND-gate output, we partially multiply out Equation (7-4) using (X + Y)(X + Z) = X + YZ: f = [c + d(a′ + b)][c′ + d′(a + b)] (7-5) Equation (7-5) would require four levels of gates to realize; however, if we multiply out d′(a + b) and d(a′ + b), we get f = (c + a′d + bd)(c′ + ad′ + bd′) (7-6) which leads directly to a three-level AND-OR-AND circuit (Figure 7-7): FIGURE 7-7 © Cengage Learning 2014 a d′ b c′ d′ f a′ d b Three levels Seven gates 16 gate inputs c d For this particular example, the best two-level solution had an AND gate at the output (Figure 7-6), and the best three-level solution had an OR gate at the output (Figure 7-5). In general, to be sure of obtaining a minimum solution, one must find both the circuit with the AND-gate output and the one with the OR-gate output. If an expression for f′ has n levels, the complement of that expression is an n-level expression for f . Therefore, to realize f as an n-level circuit with an AND-gate output, one procedure is first to find an n-level expression for f′ with an OR operation at the output level and then complement the expression for f′. In the preceding example, factoring Equation (7-3) gives a three-level expression for f′: f′ = c′(d′ + ab′) + c(d + a′b) = c′(d′ + a)(d′ + b′) + c(d + a′)(d + b′) (7-7) Complementing Equation (7-7) gives Equation (7-6), which corresponds to the three-level AND-OR-AND circuit of Figure 7-7. 204 Unit 7 7.2 NAND and NOR Gates Until this point we have designed logic circuits using AND gates, OR gates, and inverters. Exclusive-OR and equivalence gates have also been introduced in Unit 3. In this section we will define NAND and NOR gates. Logic designers frequently use NAND and NOR gates because they are generally faster and use fewer components than AND or OR gates. As will be shown later, any logic function can be implemented using only NAND gates or only NOR gates. Figure 7-8(a) shows a three-input NAND gate. The small circle (or "bubble") at the gate output indicates inversion, so the NAND gate is equivalent to an AND gate followed by an inverter, as shown in Figure 7-8(b). A more appropriate name would be an AND-NOT gate, but we will follow common usage and call it a NAND gate. The gate output is F = (ABC)′ = A′ + B′ + C′ The output of the n-input NAND gate in Figure 7-8(c) is F = (X1X2 . . . Xn)′ = X1′ + X2′ + · · · + Xn′ (7-8) FIGURE 7-8 NAND Gates © Cengage Learning 2014 A B C A B C F (a) Three-input NAND gate F (b) NAND gate equivalent X1 X2 Xn ... The output of this gate is 1 iff one or more of its inputs are 0. F (c) n-input NAND gate Figure 7-9(a) shows a three-input NOR gate. The small circle at the gate output indicates inversion, so the NOR gate is equivalent to an OR gate followed by an inverter. A more appropriate name would be an OR-NOT gate, but we will follow common usage and call it a NOR gate. The gate output is FIGURE 7-9 NOR Gates © Cengage Learning 2014 A B C A B C F (a) Three-input NOR gate F (b) NOR gate equivalent X1 X2 Xn ... F = (A + B + C)′ = A′B′C′ F (c) n-input NOR gate The output of an n-input NOR gate, shown in Figure 7-9(c), is F = (X1 + X2 + · · · + Xn)′ = X1′X2′ . . . Xn′ (7-9) Multi-Level Gate Circuits NAND and NOR Gates 205 A set of logic operations is said to be functionally complete if any Boolean function can be expressed in terms of this set of operations. The set AND, OR, and NOT is obviously functionally complete because any function can be expressed in sum-of-products form, and a sum-of-products expression uses only the AND, OR, and NOT operations. Similarly, a set of logic gates is functionally complete if all switching functions can be realized using this set of gates. Because the set of operations AND, OR, and NOT is functionally complete, any set of logic gates which can realize AND, OR, and NOT is also functionally complete. AND and NOT are a functionally complete set of gates because OR can also be realized using AND and NOT: X′ X X′Y′ (X′Y′)′ = X + Y Y′ Y If a single gate forms a functionally complete set by itself, then any switching function can be realized using only gates of that type. The NAND gate is an example of such a gate. Because the NAND gate performs the AND operation followed by an inversion, NOT, AND, and OR can be realized using only NAND gates, as shown in Figure 7-10. Thus, any switching function can be realized using only NAND gates. An easy method for converting an AND-OR circuit to a NAND circuit is discussed in the next section. Similarly, any function can be realized using only NOR gates. FIGURE 7-10 NAND Gate Realization of NOT, AND, and OR X A X′ A (AB)′ B AB A′ (A′B′)′ = A + B © Cengage Learning 2014 B B′ The following procedure can be used to determine if a given set of gates is functionally complete. First, write out a minimum sum-of-products expression for the function realized by each gate. If no complement appears in any of these expressions, then NOT cannot be realized, and the set is not functionally complete. If a complement appears in one of the expressions, then NOT can generally be realized by an appropriate choice of inputs to the corresponding gate. (We will always assume that 0 and 1 are available as gate inputs). Next, attempt to realize AND or OR, keeping in mind that NOT is now available. Once AND or OR has been realized, the other one can always be realized using DeMorgan's laws if no more direct procedure is apparent. For example, if OR and NOT are available, AND can be realized by XY = (X′ + Y′)′ (7-10) 206 Unit 7 7.3 Design of Two-Level NAND- and NOR-Gate Circuits In this section two-level circuits realizing a function F using various combinations of NAND, NOR, AND, and OR gates are obtained by converting the switching algebra expression for F into the form matching the desired gate circuit. It is difficult to extend this approach to multiple-level circuits because it requires repeated complementation of parts of the expression for F. In Sections 7.4 and 7.5, an alternative method is developed which first realizes F in the desired form using AND and OR gates. The circuit with AND and OR gates is converted to one containing NAND or NOR gates by inserting inverters in pairs to convert each AND and OR gate to a NAND or NOR gate. This approach avoids manipulation of the expression for F and is less error prone. A two-level circuit composed of AND and OR gates is easily converted to a circuit composed of NAND gates or NOR gates. This conversion is carried out by using F = (F ′)′ and then applying DeMorgan's laws: (X1 + X2 + · · · + Xn)′ = X1′X2′ · · · Xn′ (X1X2 · · · Xn)′ = X1′ + X2′ + · · · + Xn′ (7-11) (7-12) The following example illustrates conversion of a minimum sum-of-products form to several other two-level forms: F = A + BC′ + B′CD = [(A + BC′ + B′CD)′] ′ (7-13) = [A′ · (BC′)′ · (B′CD)′] ′ (by 7-11) (7-14) = [A′ · (B′ + C) · (B + C′ + D′)] ′ (by 7-12) (7-15) = A + (B′ + C)′ + (B + C′ + D′)′ (by 7-12) (7-16) Equations (7-13), (7-14), (7-15), and (7-16) represent the AND-OR, NAND-NAND, OR-NAND, and NOR-OR forms, respectively, as shown in Figure 7-11. Rewriting Equation (7-16) in the form F = 5[A + (B′ + C)′ + (B + C′ + D′)′] ′ 6 ′ (7-17) leads to a three-level NOR-NOR-INVERT circuit. However, if we want a two-level circuit containing only NOR gates, we should start with the minimum productof-sums form for F instead of the minimum sum of products. After obtaining the minimum product of sums from a Karnaugh map, F can be written in the following two-level forms: F = (A + B + C)(A + B′ + C′)(A + C′ + D) = 5[(A + B + C)(A + B′ + C′)(A + C′ + D)] ′ 6 ′ = [(A + B + C)′ + (A + B′ + C′)′ + (A + C′ + D)′] ′ = (A′B′C′ + A′BC + A′CD′)′ = (A′B′C′)′ · (A′BC)′ · (A′CD′)′ (7-18) (by 7-12) (7-19) (by 7-11) (7-20) (by 7-11) (7-21) Multi-Level Gate Circuits NAND and NOR Gates 207 F = A + BC′ + B′CD (7-13) FIGURE 7-11 Eight Basic Forms for Two-Level Circuits B C′ B′ C D © Cengage Learning 2014 F = A + (B′ + C)′ + (B + C′ + D′)′ A F F F = [A′ • (BC′)′ • (B′CD)′]′ ANDOR (7-16) B′ C B C′ D′ A NOROR (7-14) B C′ B′ C D NANDNAND A′ F ORNAND B′ C B C′ D′ A′ F F = [A′ • (B′ + C) • (B + C′ + D′)]′ (7-15) F = (A + B + C)(A + B′ + C′)(A + C′ + D) A B C A B′ C′ A C′ D F F = (A′B′C′)′ • (A′BC)′ • (A′CD′)′ (7-21) A′ B′ C′ A′ B C A′ C D′ ORAND F (7-18) NANDAND NORNOR ANDNOR A′ B′ C′ A′ B C A′ C D′ F = (A′B′C′ + A′BC + A′CD′)′ F = [(A + B + C)′ + (A + B′ + C′)′ + (A + C′ + D)′]′ (7-19) A B C A F B′ C′ A C′ D F (7-20) 208 Unit 7 Equations (7-18), (7-19), (7-20), and (7-21) represent the OR-AND, NOR-NOR, AND-NOR, and NAND-AND forms, respectively, as shown in Figure 7-11. Twolevel AND-OR-INVERT (AND-OR-INVERT) circuits are available in integrated-circuit form. Some types of NAND gates can also realize AND-NOR circuits when the so-called wired OR connection is used. The other eight possible two-level forms (AND-AND, OR-OR, OR-NOR, AND-NAND, NAND-NOR, NOR-NAND, etc.) are degenerate in the sense that they cannot realize all switching functions. Consider, for example, the following NAND-NOR circuit: a b e F c F = [(ab)′ + (cd)′ + e] ′ = abcde′ d From this example, it is clear that the NAND-NOR form can realize only a product of literals and not a sum of products. Because NAND and NOR gates are readily available in integrated circuit form, two of the most commonly used circuit forms are the NAND-NAND and the NOR-NOR. Assuming that all variables and their complements are available as inputs, the following method can be used to realize F with NAND gates: Procedure for designing a minimum two-level NAND-NAND circuit: 1. 2. 3. Find a minimum sum-of-products expression for F. Draw the corresponding two-level AND-OR circuit. Replace all gates with NAND gates leaving the gate interconnections unchanged. If the output gate has any single literals as inputs, complement these literals. Figure 7-12 illustrates the transformation of step 3. Verification that this transformation leaves the circuit output unchanged follows. In general, F is a sum of literals (ℓ1, ℓ2, . . .) and product terms (P1, P2, . . .): F = ℓ1 + ℓ2 + · · · + P1 + P2 + · · · After applying DeMorgan's law, (a) Before transformation F x1 x2 P′1 y1 y2 P 2′ ℓ′1 ℓ′2 . . . . . . P2 ℓ1 ℓ2 . . . y1 y2 . . . P1 . . . © Cengage Learning 2014 x1 x2 . . . FIGURE 7-12 AND-OR to NAND-NAND Transformation . . . . . . F = (ℓ1′ ℓ2′ · · · P1′ P2′ · · ·)′ (b) After transformation F Multi-Level Gate Circuits NAND and NOR Gates 209 So the output OR gate is replaced with a NAND gate with inputs, ℓ1′, ℓ2′, · · ·, P1′, P2′, · · · . Because product terms P1, P2, . . . are each realized with an AND gate, P1′, P2′, . . . are each realized with a NAND gate in the transformed circuit. Assuming that all variables and their complements are available as inputs, the following method can be used to realize F with NOR gates: Procedure for designing a minimum two-level NOR-NOR circuit: 1. 2. 3. Find a minimum product-of-sums expression for F. Draw the corresponding two-level OR-AND circuit. Replace all gates with NOR gates leaving the gate interconnections unchanged. If the output gate has any single literals as inputs, complement these literals. This procedure is similar to that used for designing NAND-NAND circuits. Note, however, that for the NOR-NOR circuit, the starting point is a minimum product of sums rather than a sum of products. 7.4 Design of Multi-Level NAND- and NOR-Gate Circuits The following procedure may be used to design multi-level NAND-gate circuits: 1. 2. 3. Simplify the switching function to be realized. Design a multi-level circuit of AND and OR gates. The output gate must be OR. AND-gate outputs cannot be used as AND-gate inputs; OR-gate outputs cannot be used as OR-gate inputs. Number the levels starting with the output gate as level 1. Replace all gates with NAND gates, leaving all interconnections between gates unchanged. Leave the inputs to levels 2, 4, 6, . . . unchanged. Invert any literals which appear as inputs to levels 1, 3, 5, ... The validity of this procedure is easily proven by dividing the multi-level circuit into two-level subcircuits and applying the previous results for two-level circuits to each of the two-level subcircuits. The example of Figure 7-13 illustrates the procedure. Note that if step 2 is performed correctly, each level of the circuit will contain only AND gates or only OR gates. The procedure for the design of multi-level NOR-gate circuits is exactly the same as for NAND-gate circuits except the output gate of the circuit of AND and OR gates must be an AND gate, and all gates are replaced with NOR gates. Example F1 = a′ [b′ + c(d + e′) + f ′g′] + hi′j + k Figure 7-13 shows how the AND-OR circuit for F1 is converted to the corresponding NAND circuit. 210 Unit 7 FIGURE 7-13 Multi-Level Circuit Conversion to NAND Gates Level 5 Level 4 d e′ Level 3 Level 2 Level 1 a′ c © Cengage Learning 2014 b′ F1 k f′ g′ h i′ j (a) AND-OR network Level 5 Level 4 d′ e Level 3 Level 2 Level 1 a′ c b F1 k′ f′ g′ h i′ j (b) NAND network 7.5 Circuit Conversion Using Alternative Gate Symbols Logic designers who design complex digital systems often find it convenient to use more than one representation for a given type of gate. For example, an inverter can be represented by A A′ or A A′ In the second case, the inversion "bubble" is at the input instead of the output. Figure 7-14 shows some alternative representations for AND, OR, NAND, and NOR gates. These equivalent gate symbols are based on DeMorgan's laws. FIGURE 7-14

Alternative Gate Symbols © Cengage Learning 2014 A B AB A+B A B A B (AB)′ A B (A + B)′ AB = (A′ + B′)′ A + B = (A′B′)′ (AB)′ = A′ + B′ (A + B)′ = A′B′ (a) AND (b) OR (c) NAND (d) NOR These alternative symbols can be used to facilitate the analysis and design of NANDand NOR-gate circuits. Figure 7-15(a) shows a simple NAND-gate circuit. To analyze the circuit, we will replace the NAND gates at the first and third levels with the alternative NAND gate symbol. This eliminates the inversion bubble at the circuit output. Multi-Level Gate Circuits NAND and NOR Gates FIGURE 7-15 NAND Gate Circuit Conversion A B′ 1 C D © Cengage Learning 2014 E 2 F 4 211 Z 3 (a) NAND gate network A B′ 1 A′ + B C D E 2 [(A′ + B)C]′ F 3 4 Z = (A′ + B)C + F ′ + DE (DE)′ (b) Alternate form for NAND gate network A′ B 1 C D E 2 F′ 4 Z 3 (c) Equivalent AND-OR network In the resulting circuit (Figure 7-15(b)), inverted outputs (those with a bubble) are always connected to inverted inputs, and noninverted outputs are connected to noninverted inputs. Because two inversions in a row cancel each other out, we can easily analyze the circuit without algebraically applying DeMorgan's laws. Note, for example, that the output of gate 2 is [(A′ + B)C]′, but the term (A′ + B)C appears in the output function. We can also convert the circuit to an AND-OR circuit by simply removing the double inversions (see Figure 7-15(c)). When a single input variable is connected to an inverted input, we must also complement that variable when we remove the inversion from the gate input. For example, A in Figure 7-15(b) becomes A′ in Figure 7-15(c). The circuit of AND and OR gates shown in Figure 7-16(a) can easily be converted to a NOR-gate circuit because the output gate is an AND-gate, and AND and OR gates alternate throughout the circuit. That is, AND-gate outputs connect only to OR-gate inputs, and OR-gate outputs connect only to AND-gate inputs. To carry out conversion to NOR-gates, we first replace all of the OR and AND gates with NOR gates, as shown in Figure 7-16(b). Because each inverted gate output drives an inverted gate input, the pairs of inversions cancel. However, when an input variable drives an inverted input, we have added a single inversion, so we must complement the variable to compensate. Therefore, we have complemented C and G. The resulting NOR-gate circuit is equivalent to the original AND-OR circuit. Even if AND and OR gates do not alternate, we can still convert an AND-OR circuit to a NAND or NOR circuit, but it may be necessary to add extra inverters so that each added inversion is cancelled by another inversion. The following procedure may be used to convert to a NAND (or NOR) circuit: 1. Convert all AND gates to NAND gates by adding an inversion bubble at the output. Convert all OR gates to NAND gates by adding inversion bubbles at the 212 Unit 7 FIGURE 7-16 Conversion to NOR Gates A B′ G C Z D © Cengage Learning 2014 E F (a) Circuit with OR and AND gates Double inversion cancels A B′ C′ D Complemented input cancels inversion G′ Z E F (b) Equivalent circuit with NOR gates 2. 3. FIGURE 7-17 Conversion of AND-OR Circuit to NAND Gates inputs. (To convert to NOR, add inversion bubbles at all OR-gate outputs and all AND-gate inputs.) Whenever an inverted output drives an inverted input, no further action is needed because the two inversions cancel. Whenever a noninverted gate output drives an inverted gate input or vice versa, insert an inverter so that the bubbles will cancel. (Choose an inverter with the bubble at the input or output as required.) A B′ C D E F (a) AND-OR network © Cengage Learning 2014 Bubbles cancel A B′ C D E F (b) First step in NAND conversion Added inverter A B′ Added inverter C D′ (c) Completed conversion E′ F Multi-Level Gate Circuits NAND and NOR Gates 4. 213 Whenever a variable drives an inverted input, complement the variable (or add an inverter) so the complementation cancels the inversion at the input. In other words, if we always add bubbles (or inversions) in pairs, the function realized by the circuit will be unchanged. To illustrate the procedure we will convert Figure 7-17(a) to NANDs. First, we add bubbles to change all gates to NAND gates (Figure 7-17(b)). In four places (highlighted in blue), we have added only a single inversion. This is corrected in Figure 7-17(c) by adding two inverters and complementing two variables. Note that when an inverter is added between two gates during the conversion procedure, the number of levels in the circuit is increased by 1. This is avoided if each path through the circuit alternately passes through AND and OR gates. Similarly, if the circuit containing AND and OR gates has an output OR (AND) gate and it is converted to a circuit with NOR (NAND) gates, then it is necessary to add an inverter at the output, which also increases the number of levels by 1. Hence, if a NAND (NOR) gate circuit is desired, it is usually best to start with a circuit containing AND and OR gates that has an output OR (AND) gate. An advantage of multi-level circuits is that gate fan-in can be reduced. As an example, consider F = D′E + BCE + AB′ + AC′ (7-22) A two-level AND-OR circuit implementing F requires one 4-input OR, one 3-input AND, and three 2-input ANDs. To reduce the fan-in, F can be factored. F = A(B′ + C′) + E(D′ + BC) (7-23) The resulting four-level circuit using AND and OR gates is shown in Figure 7-18. FIGURE 7-18 Limited Fan-In Circuit B′ C′ A © Cengage Learning 2014 F B C E D′ Since the output gate is an OR, the circuit can be converted to NAND gates without increasing the number of levels; Figure 7-19 is the result. Note that the threelevel OR with inputs B′ and C′ and the four-level AND with inputs B and C both become a NAND with inputs B and C; hence, both can be replaced by the same gate. FIGURE 7-19 NAND Gate Equivalent of Figure 7-18 A B C © Cengage Learning 2014 F D E 214 Unit 7 Reducing the fan-in for some functions requires inserting inverters. The fan-in for F = ABC + D can be reduced to 2 by factoring F as F = (AB)C + D. If this is implemented using two-input NAND gates, an inverter is required and the resulting circuit has four levels. 7.6 Design of Two-Level, Multiple-Output Circuits Solution of digital design problems often requires the realization of several functions of the same variables. Although each function could be realized separately, the use of some gates in common between two or more functions sometimes leads to a more economical realization. The following example illustrates this: Design a circuit with four inputs and three outputs which realizes the functions F1(A, B, C, D) = Σ m(11, 12, 13, 14, 15) F2(A, B, C, D) = Σ m(3, 7, 11, 12, 13, 15) F3(A, B, C, D) = Σ m(3, 7, 12, 13, 14, 15) (7-24) First, each function will be realized individually. The Karnaugh maps, functions, and resulting circuit are given in Figures 7-20 and 7-21. The cost of this circuit is 9 gates and 21 gate inputs. An obvious way to simplify this circuit is to use the same gate for AB in both F1 and F3. This reduces the cost to eight gates and 19 gate inputs. (Another, but less obvious, way to simplify the circuit is possible.) Observing that the term ACD is necessary for the realization of F1 and A′CD is necessary for F3, if we replace CD in F2 by A′CD + ACD, the realization of CD is unnecessary and one gate is saved. Figure 7-22 shows the reduced circuit, which requires seven gates and 18 gate inputs. Note that F2 is realized by the expression ABC′ + A′CD + ACD, which is not a minimum sum of products, and two of the terms are not prime implicants of F2. Thus in realizing multiple-output circuits, the use of a minimum sum of prime implicants FIGURE 7-20 Karnaugh Maps for Equations (7-24) AB CD AB 00 01 11 10 CD AB 00 01 11 10 CD 00 01 11 00 1 00 1 00 1 01 1 01 1 01 1 11 1 10 1 © Cengage Learning 2014 F1 1 1 11 1 1 1 10 1 11 1 1 1 1 10 F2 F3 10 Multi-Level Gate Circuits NAND and NOR Gates FIGURE 7-21 Realization of Equations (7-24) © Cengage Learning 2014 A C D 215 F1 = AB + ACD A B A B C′ F2 = ABC′ + CD C D A′ C D F3 = A′CD + AB A B FIGURE 7-22 Multiple-Output Realization of Equations (7-24) © Cengage Learning 2014 A B F1 A C D F2 A B C′ A′ C D F3 for each function does not necessarily lead to a minimum cost solution for the circuit as a whole. When designing multiple-output circuits, you should try to minimize the total number of gates required. If several solutions require the same number of gates, the one with the minimum number of gate inputs should be chosen. The next example further illustrates the use of common terms to save gates. A four-input, three-output circuit is to be designed to realize f1 = Σ m(2, 3, 5, 7, 8, 9, 10, 11, 13, 15) f2 = Σ m(2, 3, 5, 6, 7, 10, 11, 14, 15) f3 = Σ m(6, 7, 8, 9, 13, 14, 15) (7-25) First, we plot maps for f1, f2, and f3 (Figure 7-23). If each function is minimized separately, the result is f1 = bd + b′c + ab′ f2 = c + a′bd abd 10 gates, f3 = bc + ab′c′ + % or - 25 gate inputs ac′d (7-25(a)) 216 Unit 7 FIGURE 7-23 abd © Cengage Learning 2014 ab′c′ ab cd ab 00 01 11 00 01 11 1 10 1 10 cd 1 00 ab 00 01 11 10 cd 00 01 11 1 00 1 1 1 01 1 1 1 11 1 1 1 1 11 1 1 1 1 10 1 1 1 1 10 1 1 1 1 10 1 01 1 a′bd By inspecting the maps, we can see that terms a′bd (from f2), abd (from f3), and ab′c′ (from f3) can be used in f1. If bd is replaced with a′bd + abd, then the gate needed to realize bd can be eliminated. Because m10 and m11 in f1 are already covered by b′c, ab′c′ (from f3) can be used to cover m8 and m9, and the gate needed to realize ab′ can be eliminated. The minimal solution is therefore f1 = a′bd + abd + ab′c′ + b′c f2 = c + a′bd eight gates f3 = bc + ab′c′ + abd 22 gate inputs (7-25(b)) (Terms which are used in common between two functions are underlined.) When designing multiple-output circuits, it is sometimes best not to combine a 1 with its adjacent 1′s, as illustrated in the example of Figure 7-24. The solution with the maximum number of common terms is not necessarily best, as illustrated in the example of Figure 7-25. Determination of Essential Prime Implicants for Multiple-Output Realization As a first step in determining a minimum two-level, multiple-output realization, it is often desirable to determine essential prime implicants. However, we must be careful because some of the prime implicants essential to an individual function may not be essential to the multiple-output realization. For example, in Figure 7-23, bd is an essential prime implicant of f1 (only prime implicant which covers m5), but it is not essential to the multiple-output realization. The reason that bd is not essential is that m5 also appears on the f2 map and, hence, might be covered by a term which is shared by f1 and f2. We can find prime implicants which are essential to one of the functions and to the multiple-output realization by a modification of the procedure used for the singleoutput case. In particular, when we check each 1 on the map to see if it is covered by only one prime implicant, we will only check those 1′s which do not appear on the other function maps. Thus, in Figure 7-24 we find that c′d is essential to f1 for the multiple-output realization (because of m1), but abd is not essential because m15 also 217 Multi-Level Gate Circuits NAND and NOR Gates FIGURE 7-24 © Cengage Learning 2014 ab ab 00 01 11 10 cd 00 01 ab 00 01 11 10 cd 1 00 1 1 1 11 1 1 11 1 10 00 01 01 10 1 f1 ab 00 01 11 10 cd 1 11 1 10 © Cengage Learning 2014 1 00 01 1 00 1 1 01 1 1 1 f1 10 01 11 1 10 1 1 f2 ab 1 1 cd 00 ab 00 01 11 10 1 01 11 1 1 1 (b) Solution requires an extra gate 00 01 11 10 1 11 10 cd 1 f1 ab 00 01 11 10 1 1 f2 ab cd 1 00 (a) Best solution FIGURE 7-25 00 01 11 10 cd 1 f2 (a) Solution with maximum number of common terms requires 8 gates, 26 inputs 10 00 01 11 10 1 00 1 1 01 1 1 1 1 1 1 11 11 1 cd 1 1 1 10 f1 f2 (b) Best solution requires 7 gates, 18 inputs and has no common terms appears on the f2 map. In Figure 7-25, the only minterms of f1 which do not appear on the f2 map are m2 and m5. The only prime implicant which covers m2 is a′d′; hence, a′d′ is essential to f1 in the multiple-output realization. Similarly, the only prime implicant which covers m5 is a′bc′, and a′bc′ is essential. On the f2 map, bd′ is essential. Why? Once the essential prime implicants for f1 and f2 have been looped, selection of the remaining terms to form the minimum solution is obvious in this example. The techniques for finding essential prime implicants outlined above cannot be applied in a problem such as Figure 7-23, where every minterm of f1 also appears on the f2 or f3 map. A general procedure for finding the minimum multiple output AND-OR circuit requires finding the prime implicants of not only each function but, also, of the product of all functions. If three functions f1, f2, and f3 are being realized, then the prime implicants of f1, f2, f3, f1 f2, f1 f3, f2 f3, and f1 f2 f3 are required. The optimum solution is obtained by selecting the fewest prime implicants from these prime implicants to realize f1, f2, and f3. This procedure is not discussed further in this text. 7.7 Multiple-Output NAND- and NOR-Gate Circuits The procedure given in Section 7.4 for design of single-output, multi-level NANDand NOR-gate circuits also applies to multiple-output circuits. If all of the output gates are AND gates, direct conversion to a NAND-gate circuit is possible. If all 218 Unit 7 of the output gates are OR gates, direct conversion to a NOR-gate circuit is possible. Figure 7-26 gives an example of converting a two-output circuit to NOR gates. Note that the inputs to the first and third levels of NOR gates are inverted. F1 = [(a + b′)c + d] (e′ + f) FIGURE 7-26 Multi-Level Circuit Conversion to NOR Gates Level 4 Level 3 a b′ F2 = [(a + b′)c + g′] (e′ + f)h Level 2 Level 1 d F1 c e′ f © Cengage Learning 2014 h F2 g′ (a) Network of AND and OR gates a b′ d F1 c′ e′ f h′ F2 g′ (b) NOR network Problems 7.1 Using AND and OR gates, find a minimum circuit to realize f(a, b, c, d) = m4 + m6 + m7 + m8 + m9 + m10 (a) using two-level logic (b) using three-level logic (12 gate inputs minimum) 7.2 Realize the following functions using AND and OR gates. Assume that there are no restrictions on the number of gates which can be cascaded and minimize the number of gate inputs. (a) AC′D + ADE′ + BE′ + BC′ + A′D′E′ (b) AE + BDE + BCE + BCFG + BDFG + AFG 7.3 Find eight different simplified two-level gate circuits to realize F(a, b, c, d) = a′bd + ac′d Multi-Level Gate Circuits NAND and NOR Gates 219 7.4 Find a minimum three-level NAND-gate circuit to realize F(A, B, C, D) = Σ m(5, 10, 11, 12, 13) (four gates) 7.5 Realize Z = A′D + A′C + AB′C′D′ using four NOR gates. 7.6 Realize Z = ABC + AD + C′D′ using only two-input NAND gates. Use as few gates as possible. 7.7 Realize Z = AE + BDE + BCEF using only two-input NOR gates. Use as few gates as possible. 7.8 (a) Convert the following circuit to all NAND gates, by adding bubbles and inverters where necessary. (b) Convert to all NOR gates (an inverter at the output is allowed). A′ B E C D′ Z F G′ 7.9 Find a two-level, multiple-output AND-OR gate circuit to realize the following functions. Minimize the required number of gates (six gates minimum). f1 = ac + ad + b′d and f2 = a′b′ + a′d′ + cd′ 7.10 Find a minimum two-level, multiple-output AND-OR gate circuit to realize these functions. f1(a, b, c, d) = Σ m(3, 4, 6, 9, 11) f2(a, b, c, d) = Σ m(2, 4, 8, 10, 11, 12) f3(a, b, c, d) = Σ m(3, 6, 7, 10, 11) (11 gates minimum) 7.11 Find a minimum two-level OR-AND circuit to simultaneously realize F1(a, b, c, d) = Σ m(2, 3, 8, 9, 14, 15) F2(a, b, c, d) = Σ m(0, 1, 5, 8, 9, 14, 15) (minimum solution has eight gates) 7.12 Find a minimum two-level OR-AND circuit to realize the functions given in Equations (7-25) on page 215 (nine gates minimum). 7.13 (a) Find a minimum two-level NAND-NAND circuit to realize the functions given in Equations (7-25) on page 215. (b) Find a minimum two-level NOR-NOR circuit to realize the functions given in Equations (7-25). 220 Unit 7 7.14 Using AND and OR gates, find a minimum circuit to realize f(a, b, c, d) = M0 M1 M3 M13 M14 M15 (a) using two-level logic (b) using three-level logic (12 gate inputs minimum) 7.15 Using AND and OR gates, find a minimum two-level circuit to realize (a) F = a′c + bc′d + ac′d (b) F = (b′ + c)(a + b′ + d)(a + b + c′ + d) (c) F = a′cd′ + a′bc + ad (d) F = a′b + ac + bc + bd′ 7.16 Realize the following functions using AND and OR gates. Assume that there are no restrictions on the number of gates which can be cascaded and minimize the number of gate inputs. (a) ABC′ + ACD + A′BC + A′C′D (b) ABCE + ABEF + ACD′ + ABEG + ACDE 7.17 A combinational switching circuit has four inputs (A, B, C, D) and one output (F). F = 0 iff three or four of the inputs are 0. (a) Write the maxterm expansion for F. (b) Using AND and OR gates, find a minimum three-level circuit to realize F (five gates, 12 inputs). 7.18 Find eight different simplified two-level gate circuits to realize (a) F(w, x, y, z) = (x + y′ + z)(x′ + y + z)w (b) F(a, b, c, d) = Σ m(4, 5, 8, 9, 13) 7.19 Implement f(x, y, z) = Σ m(0, 1, 3, 4, 7) as a two-level gate circuit, using a minimum number of gates. (a) Use AND gates and NAND gates. (b) Use NAND gates only. 7.20 Implement f(a, b, c, d) = Σ m(3, 4, 5, 6, 7, 11, 15) as a two-level gate circuit, using a minimum number of gates. (a) Use OR gates and NOR gates. (b) Use NOR gates only. 7.21 Realize each of the following functions as a minimum two-level NAND-gate circuit and as a minimum two-level NOR-gate circuit. (a) F(A,B,C,D) = BD′ + B′CD + A′BC + A′BC′D + B′D′ (b) f(a, b, c, d) = Π M(0, 1, 7, 9, 10, 13) · Π D(2, 6, 14, 15) (c) f(a, b, c, d) = Σ m(0, 2, 5, 10) + Σ d(3, 6, 9, 13, 14, 15) (d) F(A, B, C, D, E) = Σ m(0, 2, 4, 5, 11, 14, 16, 17, 18, 22, 23, 25, 26, 31) + Σ d(3, 19, 20, 27, 28) Multi-Level Gate Circuits NAND and NOR Gates 221 (e) F(A, B, C, D, E) = Π M(3, 4, 8, 9, 10, 11, 12, 13, 14, 16, 19, 22, 25, 27) · Π D(17, 18, 28, 29) (f) f(a, b, c, d) = Π M(1, 3, 10, 11, 13, 14, 15) · Π D(4, 6) (g) f(w, x, y, z) = Σ m(1, 2, 4, 6, 8, 9, 11, 12, 13) + Σ d(0, 7, 10, 15) 7.22 A combinational switching circuit has four inputs and one output as shown. F = 0 iff three or four of the inputs are 1. (a) Write the maxterm expansion for F. (b) Using AND and OR gates, find a minimum three-level circuit to realize F (5 gates, 12 inputs). A B C D F 7.23 Implement f(a, b, c, d) = Σ m(3, 4, 5, 6, 7, 11, 15) as a two-level gate circuit, using a minimum number of gates. (a) Use AND gates and NAND gates. (b) Use OR gates and NAND gates. (c) Use NAND gates only. 7.24 (a) Use gate equivalences to convert the circuit into a four-level circuit containing only NAND gates and a minimum number of inverters. (Assume the inputs are available only in uncomplemented form.) (b) Derive a minimum SOP expression for f . (c) By manipulating the expression for f , find a three-level circuit containing only five NAND gates and inverters. A B f C 7.25 (a) Use gate equivalences to convert the circuit of Problem 7.24 into a five-level circuit containing only NOR gates and a minimum number of inverters. (Assume the inputs are available only in uncomplemented form.) (b) Derive a minimum POS expression for f . (c) By manipulating the expression for f , find a four-level circuit containing only six NOR gates and inverters. 222 Unit 7 7.26 In the circuit, replace each NOR gate by an AND or OR gate so that the resulting circuit contains the fewest inverters possible. Assume the inputs are available in both true and complemented form. Do not replace the exclusive-OR gates. A′ H I′ B C′ W J D′ G E′ F 7.27 (a) Convert the circuit shown into a four-level circuit only containing AND and OR gates and a minimum number of inverters. (b) Derive a sum-of-products expression for f . (c) Find a circuit that realizes f ′ containing only NOR gates (no internal inverters). (Hint: Use gate conversions to convert the NAND gates in the given circuit to NOR gates.) B A C D f 7.28 f(a, b, c, d, e) = Σ m(2, 3, 6, 12, 13, 16, 17, 18, 19, 22, 24, 25, 27, 28, 29, 31) (a) Find a minimum two-level NOR-gate circuit to realize f . (b) Find a minimum three-level NOR-gate circuit to realize f . 7.29 Design a minimum three-level NOR-gate circuit to realize f = a′b′ + abd + acd 7.30 Find a minimum four-level NAND- or NOR-gate circuit to realize (a) Z = abe′f + c′e′f + d′e′f + gh (b) Z = (a′ + b′ + e + f)(c′ + a′ + b)(d′ + a′ + b)(g + h) 7.31 Implement abde′ + a′b′ + c using four NOR gates. 7.32 Implement x′yz + xvy′w′ + xvy′z′ using a three-level NAND-gate circuit. Multi-Level Gate Circuits NAND and NOR Gates 223 7.33 Design a logic circuit that has a 4-bit binary number as an input and one output. The output should be 1 iff the input is a prime number (greater than 1) or zero. (a) Use a two-level NAND-gate circuit. (b) Use a two-level NOR-gate circuit. (c) Use only two-input NAND gates. 7.34 Work Problem 7.33 for a circuit that has an output 1 iff the input is evenly divisible by 3 (0 is divisible by 3). 7.35 Realize the following functions, using only two-input NAND gates. Repeat using only two-input NOR gates. (a) F = A′BC′ + BD + AC + B′CD′ (b) F = A′CD + AB′C′D + ABD′ + BC 7.36 (a) Find a minimum circuit of two-input AND and two-input OR gates to realize F(A, B, C, D) = Σ m(0, 1, 2, 3, 4, 5, 7, 9, 11, 13, 14, 15) (b) Convert your circuit to two-input NAND gates. Add inverters where necessary. (c) Repeat (b), except convert to two-input NOR gates. 7.37 Realize Z = A [BC′ + D + E(F′ + GH)] using NOR gates. Add inverters if necessary. 7.38 Show that the function of Equation (7-22) can be realized using four 2-input NOR gates and one 3-input NOR gate. Assume the inputs are available both complemented and uncomplemented. (No inverters are required.) 7.39 F(A, B, C) equals 1 if exactly two of A, B, and C are 1. (a) Find the minimum two-level OR-AND circuit for F. (b) Find a four-level circuit for F that has six 2-input NOR gates. (c) Find the minimum two-level AND-OR circuit for F. (d) Find a three-level circuit for F that has three 2-input NAND gates and two 2-input XOR gates. 7.40 (a) Use gate conversions to convert the circuit below into one containing NAND and XOR gates. (b) Find the minimum two-level OR-AND circuit for F. (c) Find a three-level circuit for F that has five 2-input NAND gates. A B C B′ C′ F A′ 224 Unit 7 7.41 In which of the following two-level circuit forms can an arbitrary switching function be realized? Verify your answers. (Assume the inputs are available in both complemented and uncomplemented form.) (a) NOR-AND (b) NOR-OR (c) NOR-NAND (d) NOR-XOR (e) NAND-AND (f) NAND-OR (g) NAND-NOR (h) NAND-XOR 7.42 Find a minimum two-level, multiple-output AND-OR gate circuit to realize these functions (eight gates minimum). f1(a, b, c, d) = Σ m(10, 11, 12, 15) + Σ d(4, 8, 14) f2(a, b, c, d) = Σ m(0, 4, 8, 9) + Σ d(1, 10, 12) f3(a, b, c, d) = Σ m(4, 11, 13, 14, 15) + Σ d(5, 9, 12) 7.43 Repeat 7.42 for the following functions (six gates). f1(a, b, c, d) = Σ m(2, 3, 5, 6, 7, 8, 10) f2(a, b, c, d) = Σ m(0, 1, 2, 3, 5, 7, 8, 10) 7.44 Repeat 7.42 for the following functions (eight gates). f1(x, y, z) = Σ m(2, 3, 4, 5) f2(x, y, z) = Σ m(1, 3, 5, 6) f3(x, y, z) = Σ m(1, 2, 4, 5, 6) 7.45 (a) Find a minimum two-level, multiple-output OR-AND circuit to realize f1 = b′d + a′b′ + c′d and f2 = a′d′ + bc′ + bd′. (b) Realize the same functions with a minimum two-level NAND-NAND circuit. 7.46 Repeat Problem 7.45 for f1 = ac′ + b′d + c′d and f2 = b′c + a′d + cd′. 7.47 (a) Find a minimum two-level, multiple-output NAND-NAND circuit to realize f1 = Σ m(3, 6, 7, 11, 13, 14, 15) and f2 = Σ m(3, 4, 6, 11, 12, 13, 14). (b) Repeat for a minimum two-level, NOR-NOR circuit. 7.48 (a) Find a minimum two-level, multiple-output NAND-NAND circuit to realize f1 = Σ m(0, 2, 4, 6, 7, 10, 14) and f2 = Σ m(0, 1, 4, 5, 7, 10, 14). (b) Repeat for a minimum two-level, multiple-output NOR-NOR circuit. 7.49 Draw a multi-level, multiple-output circuit equivalent to Figure 7-26(a) using: (a) NAND and AND gates (b) NAND gates only (a direct conversion is not possible) UNIT Combinational Circuit Design and Simulation Using Gates 8 Objectives 1. Draw a timing diagram for a combinational circuit with gate delays. 2. Define static 0- and 1-hazards and dynamic hazards. Given a combinational circuit, find all of the static 0- and 1-hazards. For each hazard, specify the order in which the gate outputs must switch in order for the hazard to actually produce a false output. 3. Given a switching function, realize it using a two-level circuit which is free of static and dynamic hazards (for single input variable changes). 4. Design a multiple-output NAND or NOR circuit using gates with limited fan-in. 5. Explain the operation of a logic simulator that uses four-valued logic. 6. Test and debug a logic circuit design using a simulator. 225 226 Unit 8 Study Guide 1. Obtain your design problem assignment from your instructor. 2. Study Section 8.1, Review of Combinational Circuit Design. 3. Generally, it is possible to redesign a circuit which has two AND gates cascaded or two OR gates cascaded so that AND and OR gates alternate. If this is not practical, the conversion to a NAND or NOR circuit by the techniques of Section 7.4 is still possible by introducing a dummy one-input OR (AND) gate between the two AND (OR) gates. When the conversion is carried out, the dummy gate becomes an inverter. Try this technique and convert the following circuit to all NAND gates. Alternatively, you may use the procedures given in Section 7.5 to do the conversion. a b′ c d′ e f g′ 4. Study Section 8.2, Design of Circuits with Limited Gate Fan-In. (a) If a realization of a switching expression requires too many inputs on one or more gates, what should be done? (b) Assuming that all variables and their complements are available as inputs and that both AND and OR gates are available, does realizing the complement of an expression take the same number of gates and gate inputs as realizing the original expression? (c) When designing multiple-output circuits with limited gate fan-in, why is the procedure of Section 7.6 of little help? 5. (a) Study Section 8.3, Gate Delays and Timing Diagrams. Complete the timing diagram for the given circuit. Assume that the AND gate has a 30-nanosecond (ns) propagation delay and the inverter has a 20-ns delay. A A B B′ Z B B′ Z 0 20 40 60 80 100 120 t (ns) Combinational Circuit Design and Simulation Using Gates 227 (b) Work Problem 8.1. 6. Study Section 8.4, Hazards in Combinational Logic. (a) Even though all of the gates in a circuit are of the same type, each individual gate may have a different propagation delay. For example, for one type of TTL NAND gate the manufacturer specifies a minimum propagation delay of 5 ns and a maximum delay of 30 ns. Sketch the gate outputs for the following circuit when the x input changes from 1 to 0, assuming the following gate delays: (a) gate 1–5 ns (b) gate 2–20 ns (c) gate 3–10 ns x x 1 0 1 2 y1 y1 3 Z y2 y2 Z 0 10 20 30 40 50 t (ns) (b) Define static 0-hazard, static 1-hazard, and dynamic hazard. (c) Using a Karnaugh map, explain why F = a′b + ac has a 1-hazard for the input change abc = 011 to 111, but not for 011 to 010. Then explain it without using the map. (d) Explain why F = (a′ + b′)(b + c) has a 0-hazard for the input change abc = 100 to 110, but not for 100 to 000. (e) Under what condition does a sum-of-products expression represent a hazard-free, two-level AND-OR circuit? (f) Under what condition does a product-of-sums expression represent a hazard-free, two-level OR-AND circuit? (g) If a hazard-free circuit of AND and OR gates is transformed to NAND or NOR gates using the procedure given in Unit 7, why will the results be hazard-free? (h) Work Problems 8.2 and 8.3. 228 Unit 8 7. Study Section 8.5, Simulation and Testing of Logic Circuits. (a) Verify that Table 8-1 is correct. Consider both the case where the unknown value, X, is 0 and the case where it is 1. (b) The following circuit was designed to realize the function F = [A′ + B + C′D][A + B′ + (C′ + D′)(C + D)] C D′ C′ D′ C D 1 2 3 G 4 1 A′ B A 0 B′ 5 6 0 7 0 F 1 0 When a student builds the circuit in lab, he finds that when A = C = 0 and B = D = 1, the output F has the wrong value and that the gate outputs are as shown. Determine some possible causes of the incorrect output if G = 0 and if G = 1 (c) Work Problems 8.4 and 8.5. 8. Study your assigned design problem and prepare a design which meets specifications. Note that only two-, three-, and four-input NAND gates (or NOR gates as specified) and inverters are available for this project; therefore, factoring some of the equations will be necessary. Try to make an economical design by using common terms; however, do not waste time trying to get an absolute minimum solution. When counting gates, count both NAND (or NOR) gates and inverters, but do not count the inverters needed for the input variables. 9. Check your design carefully before simulating it. Test it on paper by applying some input combinations of 0′s and 1′s and tracing the signals through to make sure that the outputs are correct. If you have a CAD program such as LogicAid available, enter the truth table for your design into the computer, derive the minimum two-level equations, and compare them with your solution. 10. In designing multi-level, multiple-output circuits of the type used in the design problems in this unit, it is very difficult and time-consuming to find a minimum solution. You are not expected to find the best possible solution to these problems. All of these solutions involve some “tricks,” and it is unlikely that you could find them without trying a large number of different ways of factoring your equations. Therefore, if you already have an acceptable solution, do not waste time trying to find the minimum solution. Because integrated circuit gates are quite inexpensive, it is not good engineering practice to spend a large amount of time finding the absolute minimum solution unless a very large number of units of the same type are to be manufactured. 11. Obtain a Unit 8 supplement from your instructor and follow the instructions therein regarding simulating and testing your design. Combinational Circuit Design and Simulation Using Gates 8.1 Review of Combinational Circuit Design The first step in the design of a combinational switching circuit is usually to set up a truth table which specifies the output(s) as a function of the input variables. For n input variables this table will have 2n rows. If a given combination of values for the input variables can never occur at the circuit inputs, the corresponding output values are don’t-cares. The next step is to derive simplified algebraic expressions for the output functions using Karnaugh maps, the Quine-McCluskey method, or a similar procedure. In some cases, particularly if the number of variables is large and the number of terms is small, it may be desirable to go directly from the problem statement to algebraic equations, without writing down a truth table. The resulting equations can then be simplified algebraically. The simplified algebraic expressions are then manipulated into the proper form, depending on the type of gates to be used in realizing the circuit. The number of levels in a gate circuit is equal to the maximum number of gates through which a signal must pass when going between the input and output terminals. The minimum sum of products (or product of sums) leads directly to a minimum two-level gate circuit. However, in some applications it is desirable to increase the number of levels by factoring (or multiplying out) because this may lead to a reduction in the number of gates or gate inputs. When a circuit has two or more outputs, common terms in the output functions can often be used to reduce the total number of gates or gate inputs. If each function is minimized separately, this does not always lead to a minimum multiple-output circuit. For a two-level circuit, Karnaugh maps of the output functions can be used to find the common terms. All of the terms in the minimum multiple-output circuit will not necessarily be prime implicants of the individual functions. When designing circuits with three or more levels, looking for common terms on the Karnaugh maps may be of little value. In this case, the designer will often minimize the functions separately and, then, use ingenuity to factor the expressions in such a way to create common terms. Minimum two-level AND-OR, NAND-NAND, OR-NAND, and NOR-OR circuits can be realized using the minimum sum of products as a starting point. Minimum two-level OR-AND, NOR-NOR, AND-NOR, and NAND-AND circuits can be realized using the minimum product of sums as a starting point. Design of multi-level, 229 230 Unit 8 multiple-output NAND-gate circuits is most easily accomplished by first designing a circuit of AND and OR gates. Usually, the best starting point is the minimum sumof-products expressions for the output functions. These expressions are then factored in various ways until an economical circuit of the desired form can be found. If this circuit has an OR gate at each output and is arranged so that an AND- gate (or ORgate) output is never connected to the same type of gate, a direct conversion to a NAND-gate circuit is possible. Conversion is accomplished by replacing all of the AND and OR gates with NAND gates and then inverting any literals which appear as inputs to the first, third, fifth, . . . levels (output gates are the first level). If the AND-OR circuit has an AND-gate (or OR-gate) output connected to the same type of gate, then extra inverters must be added in the conversion process (see Section 7.5, Circuit Conversion Using Alternative Gate Symbols). Similarly, design of multi-level, multiple-output NOR-gate circuits is most easily accomplished by first designing a circuit of AND and OR gates. In this case the best starting point is usually the minimum sum-of-products expressions for the complements of the output functions. After factoring these expressions to the desired form, they are then complemented to get expressions for the output functions, and the corresponding circuit of AND and OR gates is drawn. If this circuit has an AND gate at each output, and an AND-gate (or OR-gate) output is never connected to the same type of gate, a direct conversion to a NOR-gate circuit is possible. Otherwise, extra inverters must be added in the conversion process. 8.2 Design of Circuits with Limited Gate Fan-In In practical logic design problems, the maximum number of inputs on each gate (or the fan-in) is limited. Depending on the type of gates used, this limit may be two, three, four, eight, or some other number. If a two-level realization of a circuit requires more gate inputs than allowed, factoring the logic expression to obtain a multi-level realization is necessary. Example Realize f(a, b, c, d) = Σm(0, 3, 4, 5, 8, 9, 10, 14, 15) using three-input NOR gates. cd ab 00 01 11 10 00 1 1 0 1 01 0 1 0 1 11 1 0 1 0 10 0 0 1 1 map of f : f ′ = a′b′c′d + ab′cd + abc′ + a ′bc + a ′cd′ Combinational Circuit Design and Simulation Using Gates 231 As can be seen from the preceding expression, a two-level realization requires two four-input gates and one five-input gate. The expression for f ′ is factored to reduce the maximum number of gate inputs to three and, then, it is complemented: f ′ = b′d(a′c′ + ac) + a′c(b + d′) + abc′ f = [b + d′ + (a + c)(a′ + c′)][a + c′ + b′d][a′ + b′ + c] The resulting NOR-gate circuit is shown in Figure 8-1. FIGURE 8-1 a c′ b © Cengage Learning 2014 d′ a′ b′ c a c f b d′ a′ c′ The techniques for designing two-level, multiple-output circuits given in Section 7.6 are not very effective for designing multiple-output circuits with more than two levels. Even if the two-level expressions had common terms, most of these common terms would be lost when the expressions were factored. Therefore, when designing multiple-output circuits with more than two levels, it is usually best to minimize each function separately. The resulting two-level expressions must then be factored to increase the number of levels. This factoring should be done in such a way as to introduce common terms wherever possible. Example Realize the functions given in Figure 8-2, using only two-input NAND gates and inverters. If we minimize each function separately, the result is f1 = b′c′ + ab′ + a′b f2 = b′c′ + bc + a′b f3 = a′b′c + ab + bc′ FIGURE 8-2 © Cengage Learning 2014 bc a 00 bc a 0 1 1 1 00 1 01 01 0 1 1 1 a 1 11 1 10 1 1 f2 = Σ m(0, 2, 3, 4, 7) 0 1 00 01 11 f1 = Σ m(0, 2, 3, 4, 5) bc 1 11 10 1 1 f3 = Σ m(1, 2, 6, 7) Unit 8 Each function requires a three-input OR gate, so we will factor to reduce the number of gate inputs: f1 = b′(a + c′) + a′b f2 = b(a′ + c) + b′c′ f3 = a′b′c + b(a + c′) or f2 = (b′ + c)(b + c′) + a′b The second expression for f2 has a term common to f1, so we will choose the second expression. We can eliminate the remaining three-input gate from f3 by noting that a′b′c = a′(b′c) = a′(b + c′)′ Figure 8-3(a) shows the resulting circuit, using common terms a′b and a + c′. Because each output gate is an OR, the conversion to NAND gates, as shown in Figure 8-3(b), is straightforward. FIGURE 8-3 Realization of Figure 8-2 a c′ b′ © Cengage Learning 2014 232 c b c′ b′ f1 a′ b f2 b′c f3 a′ b a′ c b c′ b′ c b′ f1 a′ b f2 b′c f3 a′ b (a) (b) 8.3 Gate Delays and Timing Diagrams When the input to a logic gate is changed, the output will not change instantaneously. The transistors or other switching elements within the gate take a finite time to react to a change in input, so that the change in the gate output is delayed with respect to the input change. Figure 8-4 shows possible input and output waveforms for an inverter. If the change in output is delayed by time, ε , with respect to the input, we say that this gate has a propagation delay of ε. In practice, the propagation delay for a 0 to 1 output change may be different than the delay for a 1 to 0 change. Propagation delays for integrated circuit gates may be as short as a few nanoseconds (1 nanosecond = 10 −9 second), and in many cases these delays can be neglected. However, in the analysis of some types of sequential circuits, even short delays may be important. Timing diagrams are frequently used in the analysis of sequential circuits. These diagrams show various signals in the circuit as a function of time. Several variables are usually plotted with the same time scale so that the times at which these variables change with respect to each other can easily be observed. Combinational Circuit Design and Simulation Using Gates FIGURE 8-4 Propagation Delay in an Inverter 233 X © Cengage Learning 2014 Time X X′ X′ Time ε1 ε2 Figure 8-5 shows the timing diagram for a circuit with two gates. We will assume that each gate has a propagation delay of 20 ns (nanoseconds). This timing diagram indicates what happens when gate inputs B and C are held at constant values 1 and 0, respectively, and input A is changed to 1 at t = 40 ns and then changed back to 0 at t = 100 ns. The output of gate G1 changes 20 ns after A changes, and the output of gate G2 changes 20 ns after G1 changes. Figure 8-6 shows a timing diagram for a circuit with an added delay element. The input X consists of two pulses, the first of which is 2 microseconds (2 × 10 −6 second) wide and the second is 3 microseconds wide. The delay element has an output Y FIGURE 8-5 Timing Diagram for AND-NOR Circuit © Cengage Learning 2014 A G1 G1 A B=1 G2 C =0 20 ns 20 ns G2 0 20 40 60 80 100 120 140 t (ns) 20 ns 20 ns which is the same as the input except that it is delayed by 1 microsecond. That is, Y changes to a 1 value 1 microsecond after the rising edge of the X pulse and returns to 0 1 microsecond after the falling edge of the X pulse. The output (Z) of the AND gate should be 1 during the time interval in which both X and Y are 1. If we assume a small propagation delay in the AND gate (ε), then Z will be as shown in Figure 8-6. FIGURE 8-6 Timing Diagram for Circuit with Delay Rising edge Falling edge 3 μs © Cengage Learning 2014 2 μs 1 X 0 X 1 μs Delay Y Z 1 Y 1 Z 0 1 μs 1 μss 0 ε 0 1 2 3 4 5 6 7 Time (microseconds) 8 9 10 234 Unit 8 8.4 Hazards in Combinational Logic When the input to a combinational circuit changes, unwanted switching transients may appear in the output. These transients occur when different paths from input to output have different propagation delays. If, in response to any single input change and for some combination of propagation delays, a circuit output may momentarily go to 0 when it should remain a constant 1, we say that the circuit has a static 1-hazard. Similarly, if the output may momentarily go to 1 when it should remain a 0, we say that the circuit has a static 0-hazard. If, when the output is supposed to change from 0 to 1 (or 1 to 0), the output may change three or more times, we say that the circuit has a dynamic hazard. Figure 8-7 shows possible outputs from a circuit with hazards. In each case the steady-state output of the circuit is correct, but a switching transient appears at the circuit output when the input is changed. Types of Hazards FIGURE 8-7 © Cengage Learning 2014 1 1 0 (a) Static 1-hazard 1 0 1 1 0 (b) Static 0-hazard 0 1 0 1 0 0 (c) Dynamic hazards Figure 8-8(a) illustrates a circuit with a static 1-hazard. If A = C = 1, then F = B + B′ = 1, so the F output should remain a constant 1 when B changes from 1 to 0. However, as shown in Figure 8-8(b), if each gate has a propagation delay of 10 ns, E will go to 0 before D goes to 1, resulting in a momentary 0 (a glitch caused by the 1-hazard) appearing at the output F. Note that right after B changes to 0, both the inverter input (B) and output (B′) are 0 until the propagation delay has elapsed. During this period, both terms in the equation for F are 0, so F momentarily goes to 0. Note that hazards are properties of the circuit and are independent of the delays existing in the circuit. If the circuit is free of hazards, then for any combination of delays that might exist in the circuit and for any single input change, the output will not contain a transient. On the other hand, if a circuit contains a hazard, then there is some combination of delays and some input change for which the circuit output contains a transient. The combination of delays that produces the transient may or may not be likely to occur in an implementation of the circuit; in some cases it is very unlikely that such delays would occur. Besides depending on the delays existing in a circuit, the occurrence of transients depends on how gates respond to input changes. In some cases, if multiple input changes to a gate occur within a short time period, a gate may not respond to the input changes. For example, in Figure 8-8 assume the inverter has a delay of 2 ns rather than 10 ns. Then the D and E changes reaching the output OR gate are 2 ns apart, in which case the OR gate may not generate the 0 glitch. A gate exhibiting Combinational Circuit Design and Simulation Using Gates FIGURE 8-8 Detection of a 1-Hazard © Cengage Learning 2014 A A BC 0 1 00 0 1 01 0 1 11 1 1 10 0 0 D B F 235 1-hazard E C F = AB′ + BC (a) Circuit with a static 1-hazard B D E F 0 ns 10 ns 20 ns 30 ns 40 ns 50 ns 60 ns (b) Timing chart this behavior is said to have an inertial delay. Quite often the inertial delay value is assumed to be the same as the propagation delay of the gate; if this is the case, the circuit of Figure 8-8 will generate the 0 glitch only for inverter delays greater than 10 ns. In contrast, if a gate always responds to input changes (with a propagation delay), no matter how closely spaced the input changes may be, the gate is said to have an ideal or transport delay. If the OR gate in Figure 8-8 has an ideal delay, then the 0 glitch would be generated for any nonzero value of the inverter delay. (Inertial and transport delay models are discussed more in Unit 10.) Unless otherwise noted, the examples and problems in this unit assume that gates have an ideal delay. Hazards can be detected using a Karnaugh map (Figure 8-8(a)). As seen on the map, no loop covers both minterms ABC and AB′C. So if A = C = 1 and B changes, both terms can momentarily go to 0, resulting in a glitch in F. We can detect hazards in a two-level AND-OR circuit, using the following procedure: 1. 2. 3. Write down the sum-of-products expression for the circuit. Plot each term on the map and loop it. If any two adjacent 1′s are not covered by the same loop, a 1-hazard exists for the transition between the two 1′s. For an n-variable map, this transition occurs when one variable changes and the other n − 1 variables are held constant. If we add a loop to the map of Figure 8-8(a) and, then, add the corresponding gate to the circuit (Figure 8-9), this eliminates the hazard. The term AC remains 1 while B is changing, so no glitch can appear in the output. Note that F is no longer a minimum sum of products. 236 Unit 8 FIGURE 8-9 Circuit with Hazard Removed A © Cengage Learning 2014 B A BC 0 1 00 0 1 01 0 1 11 1 1 10 0 0 F C A F = AB′ + BC + AC Figure 8-10(a) shows a circuit with several 0-hazards. The product-of-sums representation for the circuit output is F = (A + C)(A′ + D′)(B′ + C′ + D) The Karnaugh map for this function (Figure 8-10(b)) shows four pairs of adjacent 0′s that are not covered by a common loop as indicated by the arrows. Each of these pairs corresponds to a 0-hazard. For example, when A = 0, B = 1, D = 0, and C changes from 0 to 1, a spike may appear at the Z output for some combination of gate delays. The timing diagram of Figure 8-10(c) illustrates this, assuming gate delays of 3 ns for each inverter and of 5 ns for each AND gate and each OR gate. FIGURE 8-10 Detection of a Static 0-Hazard © Cengage Learning 2014 at 5 ns, 0 →1 C 1 A at 10 ns, 0 → 1 W 2 D at 15 ns, 0→1 at 18 ns, 1→ 0 Z 4 AB 00 CD 01 00 0 0 01 0 0 11 B 3 X at 8 ns, 1→ 0 Y at 13 ns, 1→ 0 0 10 (a) Circuit with a static 0-hazard W X Y Z 5 8 10 0 0 0 0 0 (b) Karnaugh map for circuit of (a) C 0 11 10 13 15 18 20 (c) Timing diagram illustrating 0-hazard of (a) Combinational Circuit Design and Simulation Using Gates 237 We can eliminate the 0-hazards by looping additional prime implicants that cover the adjacent 0′s that are not already covered by a common loop. This requires three additional loops as shown in Figure 8-11. The resulting equation is F = (A + C)(A′ + D′)(B′ + C′ + D)(C + D′)(A + B′ + D)(A′ + B′ + C′) and the resulting circuit requires seven gates in addition to the inverters. FIGURE 8-11 Karnaugh Map Removing Hazards of Figure 8-10 © Cengage Learning 2014 AB 00 01 00 0 0 01 0 0 CD 11 10 0 11 10 0 0 0 0 0 0 Hazards in circuits with more than two levels can be determined by deriving either a SOP or POS expression for the circuit that represents a two-level circuit containing the same hazards as the original circuit. The SOP or POS expression is derived in the normal manner except that the complementation laws are not used, i.e., xx′ = 0 and x + x′ = 1 are not used. Consequently, the resulting SOP (POS) expression may contain products (sums) of the form xx′α (x + x′ + β). (α is a product of literals or it may be null; β is a sum of literals or it may be empty.) The complementation laws are not used because we are analyzing the circuit behavior resulting from an input change. As that input change propagates through the circuit, at a given point in time a line tending toward the value x may not have the value that is the complement of a line tending toward the value x′. In the SOP expression, a product of the form xx′α represents a pseudo gate that may temporarily have the output value 1 as x changes and if α = 1. Given the SOP expression, the circuit is analyzed for static 1-hazards the same as for a two-level AND-OR circuit, i.e., the products are mapped on a Karnaugh map and if two 1′s are adjacent on the map and not included in one of the products, they correspond to a static 1-hazard. The circuit can have a static 0-hazard or a dynamic hazard only if the SOP expression contains a term of the form xx′α. A static 0-hazard exists if there are two adjacent 0′s on the Karnaugh map for which α = 1 and the two input combinations differ just in the value of x. A dynamic hazard exists if there is a term of the form xx′α and two conditions are satisfied: (1) There are adjacent input combinations on the Karnaugh map differing in the value of x, with α = 1 and with opposite function values, and (2) for these input combinations the change in x propagates over at least three paths through the circuit. 238 Unit 8 As an example consider the circuit of Figure 7-7. The expression for the circuit output is f = (c′ + ad′ + bd′)(c + a′d + bd) = cc′ + acd′ + bcd′ + a′c′d + aa′dd′ + a′bdd′ + bc′d + abdd′ + bdd′ = cc′ + acd′ + bcd′ + a′c′d + aa′dd′ + bc′d + bdd′ The Karnaugh map for this function is shown as the circled 1′s in Figure 7-3. It is derived in the normal way ignoring the product terms containing both a variable and its complement. The circuit does not contain any static 1-hazards because each pair of adjacent 1′s are covered by one of the product terms. Potentially, the terms cc′ and bdd′ may cause either static 0- or dynamic hazards or both; the first for c changing and the second for d changing. (The term aa′dd′ cannot cause either hazard because, for example, if a changes the dd′ part of the product forces it to 0.) With a = 0, b = 0, and d = 0 and c changing, the circuit output is 0 before and after the change, and because the cc′ term can cause the output to temporarily become 1, this transition is a static 0-hazard. Similarly, a change in c, with a = 1, b = 0, and d = 1, is a static 0-hazard. The cc′ term cannot cause a dynamic hazard because there are only two physical paths from input c to the circuit output. The term bdd′ can cause a static 0- or dynamic hazard only if b = 1. From the Karnaugh map, it is seen that, with b = 1 and d changing, the circuit output changes for any combination of a and c, so the only possibility is that of a dynamic hazard. There are four physical paths from d to the circuit output, so a dynamic hazard exists if a d change can propagate over at least three of those paths. However, this cannot happen because, with c = 0, propagation over the upper two paths is blocked at the upper OR gate because c′ = 1 forces the OR gate output to be 1, and with c = 1 propagation over the lower two paths is blocked at the lower OR gate. The circuit does not contain a dynamic hazard. Another approach to finding the hazards is as follows: If we factor the original expression for the circuit output (without using the complementation laws), we get f = (c′ + a + b)(c′ + d′)(c + a′ + b)(c + d) Plotting the 0′s of f from this expression on a Karnaugh map reveals that there are 0-hazards when a = b = d = 0 and c changes, and also when b = 0, a = d = 1, and c changes. An expression of the form x + x′ does not appear in any sum term of f, and this indicates that there are no 1-hazards or dynamic hazards. As another example of finding static and dynamic hazards from a SOP expression, consider the circuit of Figure 8-12(a). The SOP expression for f is f = (A′C′ + B′C) (C + D) = A′CC′ + A′C′D + B′C The Karnaugh map for f in Figure 8-12(b) shows that f = 1 for the input combinations (A, B, C, D) = (0, 0, 0, 1) and (0, 0, 1, 1) and neither product of f covers these two minterms; hence, these two input combinations imply a static 1-hazard for C changing. The product A′CC′ in f indicates the possibility of a static 0-hazard and a dynamic hazard for A = 0 and C changing. The Karnaugh map shows that when f = 0, Combinational Circuit Design and Simulation Using Gates f FIGURE 8-12 Hazard Example © Cengage Learning 2014 A f C B D (a) 239 AB CD 00 01 11 10 00 0 0 0 0 01 1 1 0 0 11 1 0 0 1 10 1 0 0 1 (b) the two input combinations (0, 1, 0, 0) and (0, 1, 1, 0) meet these conditions and, hence, they imply a static 0-hazard. The Karnaugh map shows two pairs of input combinations with f changing for A = 0 and C changing—namely, (0, 0, 0, 0), (0, 0, 1, 0), and (0, 1, 0, 1), (0, 1, 1, 1). In order for these to be dynamic hazards, the C change must propagate over three or more paths to the output. The circuit shows that propagation over the three paths requires B = 0 and D = 0 as well as A = 0; thus, a dynamic hazard only occurs for (0, 0, 0, 0) and (0, 0, 1, 0). For (0, 1, 0, 1) and (0, 1, 1, 1), the C change only propagates over one path, and f can only change once. To design a circuit which is free of static and dynamic hazards, the following procedure may be used: 1. 2. Find a sum-of-products expression (F t) for the output in which every pair of adjacent 1′s is covered by a 1-term. (The sum of all prime implicants will always satisfy this condition.) A two-level AND-OR circuit based on this F t will be free of 1-, 0-, and dynamic hazards. If a different form of the circuit is desired, manipulate F t to the desired form by simple factoring, DeMorgan’s laws, etc. Treat each xi and x′i as independent variables to prevent introduction of hazards. Alternatively, you can start with a product-of-sums expression in which every pair of adjacent 0′s is covered by a 0-term, and follow the dual procedure to design a hazard-free two-level OR-AND circuit. It should be emphasized that the discussion of hazards and the possibility of resulting glitches in this section has assumed that only a single input can change at a time and that no other input will change until the circuit has stabilized. If more than one input can change at one time, then nearly all circuits will contain hazards, and they cannot be eliminated by modifying the circuit implementation. The circuit corresponding to the Karnaugh map of Figure 8-11 illustrates this. Consider the input change (A, B, C, D) = (0, 1, 0, 1) to (0, 1, 1, 0) with both C and D changing. The output is 0 before the change and will be 0 after the circuit has stabilized; however, if the C change propagates through the circuit before the D change, then the circuit will output a transient 1. Effectively, the input combination to the circuit can temporarily become (A, B, C, D) = (0, 1, 1, 1), and the circuit output will temporarily become 1 no matter how it is implemented. 240 Unit 8 Glitches are of most importance in asynchronous sequential circuits. The latches and flip-flops discussed in Unit 11 are the most important examples of asynchronous sequential circuits. Although more than one input can change at the same time for some of these circuits, restrictions are placed on the changes so that it is necessary to analyze the circuits for hazards only when a single input changes. Consequently, the discussion in this section is relevant to this important class of circuits. 8.5 Simulation and Testing of Logic Circuits An important part of the logic design process is verifying that the final design is correct and debugging the design if necessary. Logic circuits may be tested either by actually building them or by simulating them on a computer. Simulation is generally easier, faster, and more economical. As logic circuits become more and more complex, it is very important to simulate a design before actually building it. This is particularly true when the design is built in integrated circuit form, because fabricating an integrated circuit may take a long time and correcting errors may be very expensive. Simulation is done for several reasons, including (1) verification that the design is logically correct, (2) verification that the timing of the logic signals is correct, and (3) simulation of faulty components in the circuit as an aid to finding tests for the circuit. To use a computer program for simulating logic circuits, you must first specify the circuit components and connections; then, specify the circuit inputs; and, finally, observe the circuit outputs. The circuit description may be input into a simulator in the form of a list of connections between the gates and other logic elements in the circuit, or the description may be in the form of a logic diagram drawn on a computer screen. Most modern logic simulators use the latter approach. A typical simulator which runs on a personal computer uses switches or input boxes to specify the inputs and probes to read the logic outputs. Alternatively, the inputs and outputs may be specified as sequences of 0′s and 1′s or in the form of timing diagrams. A simple simulator for combinational logic works as follows: 1. 2. 3. 4. The circuit inputs are applied to the first set of gates in the circuit, and the outputs of those gates are calculated. The outputs of the gates which changed in the previous step are fed into the next level of gate inputs. If the input to any gate has changed, then the output of that gate is calculated. Step 2 is repeated until no more changes in gate inputs occur. The circuit is then in a steady-state condition, and the outputs may be read. Steps 1 through 3 are repeated every time a circuit input changes. The two logic values, 0 and 1, are not sufficient for simulating logic circuits. At times, the value of a gate input or output may be unknown, and we will represent this unknown value by X. At other times we may have no logic signal at an input, as in the case of an open circuit when an input is not connected to any output. We use Combinational Circuit Design and Simulation Using Gates 241 the logic value Z to represent an open circuit, or high impedance (hi-Z) connection. The discussion that follows assumes we are using a four-valued logic simulator with logic values 0, 1, X (unknown), and Z (hi-Z). Figure 8-13(a) shows a typical simulation screen on a personal computer. The switches are set to 0 or 1 for each input. The probes indicate the value of each gate output. In Figure 8-13(b), one gate has no connection to one of its inputs. Because that gate has a 1 input and a hi-Z input, we do not know what the hardware will do, and the gate output is unknown. This is indicated by an X in the probe. FIGURE 8-13 © Cengage Learning 2014 1 0 1 0 1 0 1 0 1 0 1 0 1 0 Probe 1 0 1 1 X 0 1 0 1 0 1 X Z 1 (b) Simulation screen with missing gate input (a) Simulation screen showing switches Table 8-1 shows AND and OR functions for four-valued logic simulation. These functions are defined in a manner similar to the way real gates work. For an AND gate, if one of the inputs is 0, the output is always 0 regardless of the other input. If one input is 1 and the other input is X (we do not know what the other input is), then the output is X (we do not know what the output is). If one input is 1 and the other input is Z (it has no logic signal), then the output is X (we do not know what the hardware will do). For an OR gate, if one of the inputs is 1, the output is 1 regardless of the other input. If one input is 0 and the other input is X or Z, the output is unknown. For gates with more than two inputs, the operations may be applied several times. TABLE 8-1 AND and OR Functions for Four-Valued Simulation · 0 1 X Z + 0 1 X Z 0 1 X Z 0 0 0 0 0 1 X X X 0 X X X 0 X X X 0 1 X Z 0 1 X X 1 1 1 1 X 1 X X X 1 X X © Cengage Learning 2014 A combinational logic circuit with a small number of inputs may easily be tested with a simulator or in lab by checking the circuit outputs for all possible combinations of the input values. When the number of inputs is large, it is usually possible to find a relatively small set of input test patterns which will test for all possible faulty gates in the circuit.1 1 Methods for test pattern generation are described in Alexander Miczo, Digital Logic Testing and Simulation, 2nd ed. (John Wiley & Sons, 2003). 242 Unit 8 If a circuit output is wrong for some set of input values, this may be due to several possible causes: 1. 2. 3. Incorrect design Gates connected wrong Wrong input signals to the circuit If the circuit is built in lab, other possible causes include 4. 5. Defective gates Defective connecting wires Fortunately, if the output of a combinational logic circuit is wrong, it is very easy to locate the problem systematically by starting at the output and working back through the circuit until the trouble is located. For example, if the output gate has the wrong output and its inputs are correct, this indicates that the gate is defective. On the other hand, if one of the inputs is wrong, then either the gate is connected wrong, the gate driving this input has the wrong output, or the input connection is defective. Example The function F = AB(C′D + CD′) + A′B′(C + D) is realized by the circuit of Figure 8-14: FIGURE 8-14 Logic Circuit with Incorrect Output C′ © Cengage Learning 2014 D C D′ 1 2 0 3 0 C D 4 1 A B A′ 1 B′ 5 6 1 7 1 F 0 A student builds the circuit in a lab and finds that when A = B = C = D = 1, the output F has the wrong value, and that the gate outputs are as shown in Figure 8-14. The reason for the incorrect value of F can be determined as follows: 1. 2. 3. 4. The output of gate 7 (F) is wrong, but this wrong output is consistent with the inputs to gate 7, that is, 1 + 0 = 1. Therefore, one of the inputs to gate 7 must be wrong. In order for gate 7 to have the correct output (F = 0), both inputs must be 0. Therefore, the output of gate 5 is wrong. However, the output of gate 5 is consistent with its inputs because 1 · 1 · 1 = 1. Therefore, one of the inputs to gate 5 must be wrong. Either the output of gate 3 is wrong, or the A or B input to gate 5 is wrong. Because C′D + CD′ = 0, the output of gate 3 is wrong. The output of gate 3 is not consistent with the outputs of gates 1 and 2 because 0 + 0 ≠ 1. Therefore, either one of the inputs to gate 3 is connected wrong, gate 3 is defective, or one of the input connections to gate 3 is defective. This example illustrates how to troubleshoot a logic circuit by starting at the output gate and working back until the wrong connection or defective gate is located. Combinational Circuit Design and Simulation Using Gates 243 Problems 8.1 Complete the timing diagram for the given circuit. Assume that both gates have a propagation delay of 5 ns. W X W V Y Z X Y V Z 0 5 10 15 20 25 30 35 40 t (ns) 8.2 Consider the following logic function. F (A, B, C, D) = Σ m (0, 4, 5, 10, 11, 13, 14, 15) (a) Find two different minimum circuits which implement F using AND and OR gates. Identify two hazards in each circuit. (b) Find an AND-OR circuit for F which has no hazards. (c) Find an OR-AND circuit for F which has no hazards. 8.3 For the following circuit: B E G C F A D (a) Assume that the inverters have a delay of 1 ns and the other gates have a delay of 2 ns. Initially A = 0 and B = C = D = 1, and C changes to 0 at time = 2 ns. Draw a timing diagram and identify the transient that occurs. (b) Modify the circuit to eliminate the hazard. 8.4 Using four-valued logic, find A, B, C, D, E, F, G, and H. 1 A C E G (no connection) D B F H 244 Unit 8 8.5 The circuit below was designed to implement the logic equation F = AB′D + BC′D′ + BCD, but it is not working properly. The input wires to gates 1, 2, and 3 are so tightly packed, it would take you a while to trace them all back to see whether the inputs are correct. It would be nice to only have to trace whichever one is incorrectly wired. When A = B = 0 and C = D = 1, the inputs and outputs of gate 4 are as shown. Is gate 4 working properly? If so, which of the other gates either is connected incorrectly or is malfunctioning? A 1 B Mess of Wires C 1 1 2 4 1 F 0 3 D 8.6 (a) Assume the inverters have a delay of 1 ns and the other gates have a delay of 2 ns. Initially A = B = C = 0 and D = 1; C changes to 1 at time 2 ns. Draw a timing diagram showing the glitch corresponding to the hazard. (b) Modify the circuit so that it is hazard free. (Leave the circuit as a two-level, OR- AND circuit.) A D C B E F H G 8.7 A two-level, NOR-NOR circuit implements the function f(a, b, c, d) = (a + d′)(b′ + c + d)(a′ + c′ + d′)(b′ + c′ + d). (a) Find all hazards in the circuit. (b) Redesign the circuit as a two-level, NOR-NOR circuit free of all hazards and using a minimum number of gates. 8.8 F(A, B, C, D) = Σ m(0, 2, 3, 5, 6, 7, 8, 9, 13, 15) (a) Find three different minimum AND-OR circuits that implement F. Identify two hazards in each circuit. Then find an AND-OR circuit for F that has no hazards. (b) There are two minimum OR-AND circuits for F ; each has one hazard. Identify the hazard in each circuit, and then find an OR-AND circuit for F that has no hazards. Combinational Circuit Design and Simulation Using Gates 245 8.9 Consider the following three-level NOR circuit: (a) Find all hazards in this circuit. (b) Redesign the circuit as a three-level NOR circuit that is free of all hazards. A B C f D 8.10 Draw the timing diagram for V and Z for the circuit. Assume that the AND gate has a delay of 10 ns and the OR gate has a delay of 5 ns. W X V 10 ns 5 ns Y Z W X Y V Z 0 5 10 15 20 25 30 35 40 45 50 55 t (ns) 8.11 Consider the three-level circuit corresponding to the expression f(A, B, C, D) = (A + B)(B′C′ + BD′). (a) Find all hazards in this circuit. (b) Redesign the circuit as a three-level NOR circuit that is free of all hazards. 8.12 Complete the timing diagram for the given circuit. Assume that both gates have a propagation delay of 5 ns. W W X Y X V Z Y V Z 0 5 10 15 20 25 30 35 40 t (ns) 246 Unit 8 8.13 Implement the logic function from Figure 8.10(b) as a minimum sum of products. Find the static hazards and tell what minterms they are between. Implement the same logic function as a sum of products without any hazards. 8.14 Using four-valued logic, find A, B, C, D, E, F, G, and H. C (no connection) A D F H 0 B G (no connection) E 8.15 The following circuit was designed to implement the logic equation F = (A + B′ + C′) (A′ + B + C′)(A′ + B′ + C), but it is not working properly. The input wires to gates 1, 2, and 3 are so tightly packed, it would take you a while to trace them all back to see whether the inputs are correct. It would be nice to only have to trace whichever one is incorrectly wired. When A = B = C = 1, the inputs and outputs of gate 4 are as shown. Is gate 4 working properly? If so, which of the other gates either is connected incorrectly or is malfunctioning? A B 1 Mess of Wires C 2 1 0 4 0 F 0 3 8.16 Consider the following logic function. F(A, B, C, D) = Σ m(0, 2, 5, 6, 7, 8, 9, 12, 13, 15) (a) Find two different minimum AND-OR circuits which implement F. Identify two hazards in each circuit. Then find an AND-OR circuit for F that has no hazards. (b) The minimum OR-AND circuit for F has one hazard. Identify it, and then find an OR-AND circuit for F that has no hazards. Design Problems Seven-Segment Indicator Several of the problems involve the design of a circuit to drive a seven-segment indicator (see Figure 8-15). The seven-segment indicator can be used to display any one of the decimal digits 0 through 9. For example, “1” is displayed by lighting segments Combinational Circuit Design and Simulation Using Gates 247 2 and 3, “2” by lighting segments 1, 2, 7, 5, and 4, and “8” by lighting all seven segments. A segment is lighted when a logic 1 is applied to the corresponding input on the display module. FIGURE 8-15 Circuit Driving Seven-Segment Module © Cengage Learning 2014 Seven-Segment Indicator A Inputs From B Toggle C Switches D Circuit to be Designed X1 X2 X3 X4 X5 X6 X7 1 1 2 6 3 7 2 4 5 5 6 7 3 4 8.A Design an 8-4-2-1 BCD code converter to drive a seven-segment indicator. The four inputs to the converter circuit (A, B, C, and D in Figure 8-15) represent an 8-4-2-1 binary-coded-decimal digit. Assume that only input combinations representing the digits 0 through 9 can occur as inputs, so that the combinations 1010 through 1111 are don’t-cares. Design your circuit using only two-, three-, and four-input NAND gates and inverters. Try to minimize the number of gates required. The variables A, B, C, and D will be available from toggle switches. Use (not) for 6. Use (not) for 9. Any solution that uses 18 or fewer gates and inverters (not counting the four inverters for the inputs) is acceptable. 8.B Design an excess-3 code converter to drive a seven-segment indicator. The four inputs to the converter circuit (A, B, C, and D in Figure 8-15) represent an excess-3 coded decimal digit. Assume that only input combinations representing the digits 0 through 9 can occur as inputs, so that the six unused combinations are don’tcares. Design your circuit using only two-, three-, and four-input NAND gates and inverters. Try to minimize the number of gates and inverters required. The variables A, B, C, and D will be available from toggle switches. Use (not) for 6. Use (not) for 9. Any solution with 16 or fewer gates and inverters (not counting the four inverters for the inputs) is acceptable. 8.C Design a circuit which will yield the product of two binary numbers, n2 and m2, where 002 ≤ n2 ≤ 112 and 0002 ≤ m2 ≤ 1012. For example, if n2 = 102 and m2 = 0012, then the product is n2 × m2 = 102 × 0012 = 00102. Let the variables A and B represent the first and second digits of n2, respectively (i.e., in this example A = 1 and B = 0). Let the variables C, D, and E represent the first, second, and third digits of 248 Unit 8 m2, respectively (in this example C = 0, D = 0, and E = 1). Also let the variables W, X, Y, and Z represent the first, second, third, and fourth digits of the product. (In this example W = 0, X = 0, Y = 1, and Z = 0.) Assume that m2 > 1012 never occurs as a circuit input. n 2 Input m2 Input A B C D E W Circuit to be Designed X Y Product of n2 × m2 Z Design the circuit using only two-, three-, and four-input NOR gates and inverters. Try to minimize the total number of gates and inverters required. The variables A, B, C, D, and E will be available from toggle switches. Any solution that uses 15 or fewer gates and inverters (not counting the five inverters for the inputs) is acceptable. 8.D Work Design Problem 8.C using two-, three-, and four-input NAND gates and inverters instead of NOR gates and inverters. Any solution that uses 14 gates and inverters or less (not counting the five inverters for the inputs) is acceptable. 8.E Design a circuit which multiplies two 2-bit binary numbers and displays the answer in decimal on a seven-segment indicator. In Figure 8-15, A and B are two bits of a binary number N1, and C and D are two bits of a binary number N2. The product (N1 × N2) is to be displayed in decimal by lighting appropriate segments of the seven-segment indicator. For example, if A = 1, B = 0, C = 1, and D = 0, the number “4” is displayed by lighting segments 2, 3, 6, and 7. Use (not) for 6. Use (not) for 9. Design your circuit using only two-, three-, and four-input NAND gates and inverters. Try to minimize the number of gates required. The variables A, B, C, and D will be available from toggle switches. Any solution that uses 18 or fewer gates and inverters (not counting the four inverters for the inputs) is acceptable. 8.F Design a Gray code converter to drive a seven-segment indicator. The four inputs to the converter circuit (A, B, C, and D in Figure 8-15) represent a decimal digit coded using the Gray code of Table 1-2. Assume that only input combinations representing the digits 0 through 9 can occur as inputs, so that the six unused combinations are don’t-care terms. Design your circuit using only two-, three-, and four-input NAND gates and inverters. Try to minimize the numbers of gates and inverters required. The variables A, B, C, and D will be available from toggle switches. Use (not) for 6. Use (not) for 9. Any solution with 20 or fewer gates and inverters (not counting the four inverters for the inputs) is acceptable. Combinational Circuit Design and Simulation Using Gates 249 8.G Design a circuit that will add either 1 or 2 to a 4-bit binary number N. Let the inputs N3, N2, N1, N0 represent N. The input K is a control signal. The circuit should have outputs M3, M2, M1, M0, which represent the 4-bit binary number M. When K = 0, M = N + 1. When K = 1, M = N + 2. Assume that the inputs for which M > 11112 will never occur. Design the circuit using only two-, three-, and four-input NAND gates and inverters. Try to minimize the total number of gates and inverters required. The input variables K, N3, N2, N1, and N0 will be available from toggle switches. Any solution that uses 13 or fewer gates and inverters (not counting the five inverters for the inputs) is acceptable. 8.H Work Problem 8.A, except use 4-2-1-8 code instead of 8-4-2-1 code. For example, in 4-2-1-8 code, 9 is represented by 0011. Also change the representations of digits 6 and 9 to the opposite form given in Problem 8.A. Any solution with 20 or fewer gates and inverters (not counting the four inverters for the inputs) is acceptable. 8.I Work Problem 8.B, except use excess-2 code instead of excess-3 code. (In excess2 code, 0 is represented by 0010, 1 by 0011, 2 by 0100, etc.). Any solution with 17 or fewer gates and inverters (not counting the four inverters for the inputs) is acceptable. 8.J Design a circuit which will multiply a 3-bit binary number CDE by 2, 3, or 5, depending on the value of a 2-bit code AB (00, 01, or 10), to produce a 4-bit result WXYZ. If the result has a value greater than or equal to 15, WXYZ should be 1111 to indicate an overflow. Assume that the code AB = 11 will never occur. Design your circuit using only two-, three-, and four-input NOR gates and inverters. Try to minimize the number of gates required. The inputs A, B, C, D, and E will be available from toggle switches. Any solution which uses 19 or fewer gates and inverters (not counting the five inverters for the inputs) is acceptable. 8.K Design a circuit which will divide a 5-bit binary number by 3 to produce a 4-bit binary quotient. Assume that the input number is in the range 0 through 27 and that numbers in the range 28 through 31 will never occur as inputs. Design your circuit using only two-, three-, and four-input NAND gates and inverters. Try to minimize the number of gates required. The inputs A, B, C, D, and E will be available from toggle switches. Any solution which uses 22 or fewer gates and inverters (not counting the five inverters for the inputs) is acceptable. 8.L Design an excess-3 code converter to drive a seven-segment indicator. The four inputs (A, B, C, D) to the converter circuit represent an excess-3 digit. Input combinations representing the numbers 0 through 9 should be displayed as decimal digits. The input combinations 0000, 0001, and 0010 should be interpreted as an error, and an “E” should be displayed. Assume that the input combinations 1101, 1110, and 1111 will never occur. Design your circuit using only two-, three-, and four-input NOR gates and inverters. Any solution with 18 or fewer gates and inverters (not counting the four inverters for the inputs) is acceptable. 250 Unit 8 Use (not) for 6. Use (not) for 9. 8.M Design a circuit which displays the letters A through J on a seven-segment indicator. The circuit has four inputs W, X, Y, Z which represent the last 4 bits of the ASCII code for the letter to be

displayed. For example, if WXYZ = 0001, “A” will be displayed. The letters should be displayed in the following form: Design your circuit using only two-, three-, and four-input NOR gates and inverters. Any solution with 22 or fewer gates and inverters (not counting the four inverters for the inputs) is acceptable. 8.N A simple security system for two doors consists of a card reader and a keypad. Card Reader Keypad A B C D E Logic Circuit X To Door 1 Y To Door 2 Z To Alarm A person may open a particular door if he or she has a card containing the corresponding code and enters an authorized keypad code for that card. The outputs from the card reader are as follows: No card inserted Valid code for door 1 Valid code for door 2 Invalid card code A B 0 0 1 1 0 1 1 0 To unlock a door, a person must hold down the proper keys on the keypad and, then, insert the card in the reader. The authorized keypad codes for door 1 are 101 and 110, and the authorized keypad codes for door 2 are 101 and 011. If the card has an invalid code or if the wrong keypad code is entered, the alarm will ring when the card is inserted. If the correct keypad code is entered, the corresponding door will be unlocked when the card is inserted. Design the logic circuit for this simple security system. Your circuit’s inputs will consist of a card code AB, and a keypad code CDE. The circuit will have three outputs XYZ (if X or Y = 1, door 1 or 2 will be opened; if Z = 1, the alarm will sound). Design your circuit using only two-, three-, and four-input NOR gates and inverters. Any solution Combinational Circuit Design and Simulation Using Gates 251 with 19 or fewer gates and inverters (not counting the five inverters for the inputs) is acceptable. Use toggle switches for inputs A, B, C, D, and E when you test your circuit. 8.O Work Design Problem 8.A using two-, three-, and four-input NOR gates and inverters instead of NAND gates and inverters. Any solution that uses 19 gates and inverters or fewer (not counting the four inverters for the inputs) is acceptable. 8.P Work Design Problem 8.F using two-, three-, and four-input NOR gates and inverters instead of NAND gates and inverters. Any solution that uses 21 gates and inverters or fewer (not counting the four inverters for the inputs) is acceptable. 8.Q Work Design Problem 8.H using two-, three-, and four-input NOR gates and inverters instead of NAND gates and inverters. Any solution that uses 17 gates and inverters or fewer (not counting the four inverters for the inputs) is acceptable. 8.R Work Design Problem 8.I using two-, three-, and four-input NOR gates and inverters instead of NAND gates and inverters. Any solution that uses 16 gates and inverters or fewer (not counting the four inverters for the inputs) is acceptable. 8.S Design a “disk spinning” animation circuit for a CD player. The input to the circuit will be a 3-bit binary number A1A2A3 provided by another circuit. It will count from 0 to 7 in binary, and then it will repeat. (You will learn to design such counters in Unit 12.) The animation will appear on the top four lights of the LED display of Figure 8-15, i.e., on X1, X2, X7, and X6, going clockwise. The animation should consist of a blank spot on a disk spinning around once, beginning with X1. Then, the entire disk should blink on and off twice. The pattern is shown. Design your circuit using only two-, three-, and four-input NOR gates and inverters. Try to minimize the number of gates required. Any solution which uses 11 or fewer gates (not counting the four inverters for the inputs) is acceptable. UNIT Multiplexers, Decoders, and Programmable Logic Devices 9 Objectives 252 1. Explain the function of a multiplexer. Implement a multiplexer using gates. 2. Explain the operation of three-state buffers. Determine the resulting output when three-state buffer outputs are connected together. Use three-state buffers to multiplex signals onto a bus. 3. Explain the operation of a decoder and encoder. Use a decoder with added gates to implement a set of logic functions. Implement a decoder or priority encoder using gates. 4. Explain the operation of a read-only memory (ROM). Use a ROM to implement a set of logic functions. 5. Explain the operation of a programmable logic array (PLA). Use a PLA to implement a set of logic functions. Given a PLA table or an internal connection diagram for a PLA, determine the logic functions realized. 6. Explain the operation of a programmable array logic device (PAL). Determine the programming pattern required to realize a set of logic functions with a PAL. 7. Explain the operation of a complex programmable logic device (CPLD) and a field-programmable gate array (FPGA). 8. Use Shannon’s expansion theorem to decompose a switching function. Multiplexers, Decoders, and Programmable Logic Devices 253 Study Guide 1. Read Section 9.1, Introduction. 2. Study Section 9.2, Multiplexers. (a) Draw a logic circuit for a 2-to-1 multiplexer (MUX) using gates. (b) Write the equation for a 4-to-1 MUX with control inputs A and C. Z = ____________ (c) By tracing signals on Figure 9-3, determine what will happen to Z if A = 1, B = 0 and C changes from 0 to 1. (d) Use three 2-to-1 MUXes to make a 4-to-1 MUX with control inputs A and B. Draw the circuit. (Hint: One MUX should have I0 and I1 inputs, and another should have I2 and I3 inputs.) (e) Observe that if A = 0, A ⊕ B = B, and that if A = 1, A ⊕ B = B′. Using this observation, construct an exclusive-OR gate using a 2-to-1 multiplexer and one inverter. (f) Work Problems 9.1 and 9.2. 4 (g) This section introduces bus notation. The bus symbol A represents a group of four wires: A3 ______ A2 ______ A1 ______ A0 ______ 254 Unit 9 Draw the bus symbol for B2______ B1______ B0______ (h) Represent the circuit of Figure 4-3 by one 4-bit full adder with two bus inputs, one bus output, and terminals for carry input C0 and output C4. Note that the carries C3, C2, and C1 will not appear on your circuit diagram because they are signals internal to the 4-bit adder. 3. Study Section 9.3, Three-State Buffers. (a) Determine the output of each three-state buffer: 0 1 1 1 1 0 0 1 1 (b) Determine the inputs for each three-state buffer (use X if an input is a don’t-care). Z 1 0 1 (c) Determine the output for each circuit. Use X to represent an unknown output. 1 1 1 1 1 0 0 1 1 0 1 0 0 0 0 0 C (d) The symbol A control input: 2 2 represents 2 three-state buffers with a common B Multiplexers, Decoders, and Programmable Logic Devices 255 C A1 B1 A0 B0 Using bus notation, draw an equivalent circuit for: G E2 F2 E1 F1 E0 F0 (e) For the following circuit, determine the 4-bit output (P) if M = 0. ______ Repeat for M = 1. ______ 4 4 0101 M P 4 4 1100 (f) Specify the AND-gate inputs so that the given circuit is equivalent to the 4-to-1 MUX in Figure 9-2. (Z in the following figure represents an output terminal, not high impedance.) I0 I1 Z I2 I3 256 Unit 9 (g) Work Problem 9.3. 4. Study Section 9.4, Decoders and Encoders. (a) The 7442 4-to-10 line decoder (Figure 9-18) can be used as a 3-to-8 line decoder. To do this, which three lines should be used as inputs? ____________ The remaining input line should be set equal to ______. (b) Complete the following table for a 4-to-2 priority encoder: y0 y1 y2 y3 a b c What will a,b, and c be if y0 y1 y2 y3 is 0101? (c) Work Problem 9.4, 9.5, and 9.6. 5. Study Section 9.5, Read-Only Memories. (a) The following diagram shows the pattern of 0’s and 1’s stored in a ROM with eight words and four bits per word. What will be the values of F1, F2, F3, and F4 if A = 0 and B = C = 1? Give the minterm expansions for F1 and F2: A B C Decoder 0 1 0 1 1 1 0 0 F1 1 0 0 0 1 1 0 1 F2 1 1 0 1 0 1 0 0 F3 0 0 1 0 1 0 0 1 F4 F1 = F2 = (b) When asked to specify the size of a ROM, give the number of words and the number of bits per word. What size ROM is required to realize four functions of 5 variables? What size ROM is required to realize eight functions of 10 variables? Multiplexers, Decoders, and Programmable Logic Devices 257 (c) When specifying the size of a ROM, assume that you are specifying a standard size ROM with 2n words. What size ROM is required to convert 8-4-2-1 BCD code to 2-out-of-5 code? (See Table 1-2, page 22.) What size ROM would be required to realize the decoder given in Figure 9-18? (d) Draw an internal connection diagram for a ROM which would perform the same function as the circuit of Figure 7-22. (Indicate the presence of switching elements by dots at the intersection of the word lines and output lines.) (e) Explain the difference between a mask-programmable ROM and an EEPROM. Which would you use for a new design which had not yet been debugged? (f) Work Problem 9.7. 6. Study Section 9.6, Programmable Logic Devices. (a) When you are asked to specify the size of a PLA, give the number of inputs, the number of product terms, and the number of outputs. What size PLA would be required to realize Equations (7-24) if no simplification of the minterm expansions were performed? (b) If the realization of Equations (7-24) shown in Figure 7-22 were converted to a PLA realization, what size PLA would be required? (c) Specify the contents of the PLA of question (b) in tabular form. Your table should have four rows. (You will only need seven 1’s on the right side of your table. If you get eight 1’s, you are probably doing more work than is necessary.) (d) Draw an internal connection diagram for the PLA of (b). (Use X’s to indicate the presence of switching elements in the AND and OR arrays.) 258 Unit 9 (e) Given the following PLA table, plot maps for Z1,Z2, and Z3. A B C Z1 Z2 Z3 − 0 1 1 0 0 1 1 1 0 1 0 0 1 0 1 − 0 0 − − 1 1 0 1 1 0 1 0 0 0 0 0 1 1 1 A BC 0 1 0 1 0 00 00 00 01 01 01 11 11 11 10 10 10 Z1 Z2 1 Z3 (The Z1 map should have six 1’s, Z2 should have five, and Z3 should have four.) (f) For a truth table, any combination of input values will select exactly one row. Is this statement true for a PLA table? For any combination of input values, the output values from a PLA can be determined by inspection of the PLA table. Consider Table 9-1, which represents a PLA with three inputs and four outputs. If the inputs are ABC = 110, which three rows in the table are selected? In a given output column, what is the output if some of the selected rows are 1’s and some are 0’s? (Remember that the output bits for the selected rows are ORed together.) When ABC = 110, what are the values of F0F1F2F3 at the PLA output? When ABC = 010, which rows are selected and what are the values of F0F1F2F3 at the PLA output? (g) Which interconnection points in Figure 9-32(a) must be set in order to realize the function shown in Figure 9-32(b)? (h) What size of PAL could be used to realize the 8-to-1 MUX of Figure 9-3? The quad MUX of Figure 9-7? Give the number of inputs, the number of OR gates, and the maximum number of inputs to an OR gate. (i) Work Problems 9.8, 9.9, and 9.10. 7. Study Section 9.7, Complex Programmable Logic Devices. Work Problem 9.11. 8. Study Section 9.8, Field-Programmable Gate Arrays. (a) For the CLB of Figure 9-37, write a logic equation for H in terms of F, G, and H1. Multiplexers, Decoders, and Programmable Logic Devices 259 (b) How many 4-variable function generators are required to implement a four-input OR gate? A 4-variable function with 13 minterms? (c) Expand the function of Equation (9-9) about the variable c instead of a. Expand it algebraically and, then, expand it by using the Karnaugh map of Figure 9-39. (Hint: How should you split the map into two halves?) (d) Draw a diagram showing how to implement Equation (9-12) using four function generators and a 4-to-1 MUX. (e) In the worst case, how many 4-variable function generators are required to realize a 7-variable function (assume the necessary MUXes are available). (f) Show how to realize K = abcdefg using only two 4-variable function generators. (Hint: Use the output of one function generator as an input to the other.) (g) Work Problems 9.12 and 9.13. 9. When you are satisfied that you can meet all of the objectives, take the readiness test. Multiplexers, Decoders, and Programmable Logic Devices 9.1 Introduction Until this point we have mainly been concerned with basic principles of logic design. We have illustrated these principles using gates as our basic building blocks. In this unit we introduce the use of more complex integrated circuits (ICs) in logic design. Integrated circuits may be classified as small-scale integration (SSI), medium-scale integration (MSI), large-scale integration (LSI), or very-large-scale integration (VLSI), depending on the number of gates in each integrated circuit package and the type of function performed. SSI functions include NAND, NOR, AND, and OR gates, inverters, and flip-flops. SSI integrated circuit packages typically contain one to four gates, six inverters, or one or two flip-flops. MSI integrated circuits, such as adders, multiplexers, decoders, registers, and counters, perform more complex functions. Such integrated circuits typically contain the equivalent of 12 to 100 gates in one package. More complex functions such as memories and microprocessors are classified as LSI or VLSI integrated circuits. An LSI integrated circuit generally contains 100 to a few thousand gates in a single package, and a VLSI integrated circuit contains several thousand gates or more. It is generally uneconomical to design digital systems using only SSI and MSI integrated circuits. By using LSI and VLSI functions, the required number of integrated circuit packages is greatly reduced. The cost of mounting and wiring the integrated circuits as well as the cost of designing and maintaining the digital system may be significantly lower when LSI and VLSI functions are used. This unit introduces the use of multiplexers, decoders, encoders, and three-state buffers in logic design. Then read-only memories (ROMs) are described and used to implement multiple-output combinational logic circuits. Finally, other types of programmable logic devices (PLDs), including programmable logic arrays (PLAs), programmable array logic devices (PALs), complex programmable logic devices (CPLDs), and field- programmable gate arrays (FPGAs) are introduced and used in combinational logic design. 260 Multiplexers, Decoders, and Programmable Logic Devices 261 9.2 Multiplexers A multiplexer (or data selector, abbreviated as MUX) has a group of data inputs and a group of control inputs. The control inputs are used to select one of the data inputs and connect it to the output terminal. Figure 9-1 shows a 2-to-1 multiplexer and its switch analog. When the control input A is 0, the switch is in the upper position and the MUX output is Z = I0; when A is 1, the switch is in the lower position and the MUX output is Z = I1. In other words, a MUX acts like a switch that selects one of the data inputs (I0 or I1) and transmits it to the output. The logic equation for the 2-to-1 MUX is therefore: Z = A′I0 + AI1 FIGURE 9-1 2-to-1 Multiplexer and Switch Analog © Cengage Learning 2014 I0 I1 2-to-1 MUX I0 Z Z I1 A A Figure 9-2 shows diagrams for a 4-to-1 multiplexer, 8-to-1 multiplexer, and 2n-to-1 multiplexer. The 4-to-1 MUX acts like a four-position switch that transmits one of the four inputs to the output. Two control inputs (A and B) are needed to select one of the four inputs. If the control inputs are AB = 00, the output is I0; similarly, the control inputs 01, 10, and 11 give outputs of I1, I2, and I3, respectively. The 4-to-1 multiplexer is described by the equation FIGURE 9-2 Multiplexers © Cengage Learning 2014 I0 Data I1 inputs I 2 4-to-1 MUX I3 A B Control inputs Z I0 I1 I2 I3 I4 I5 I6 I7 2n data lines 8-to-1 MUX .. . Z = A′B′I0 + A′BI1 + AB′I2 + ABI3 (9-1) 2n-to-1 MUX Z Z ... n control inputs A B C Similarly, the 8-to-1 MUX selects one of eight data inputs using three control inputs. It is described by the equation Z = A′B′C′I0 + A′B′CI1 + A′BC′I2 + A′BCI3 + AB′C′I4 + AB′CI5 + ABC′I6 + ABCI7 (9-2) 262 Unit 9 When the control inputs are ABC = 011, the output is I3, and the other outputs are selected in a similar manner. Figure 9-3 shows an internal logic diagram for the 8-to-1 MUX. In general, a multiplexer with n control inputs can be used to select any one of 2n data inputs. The general equation for the output of a MUX with n control inputs and 2n data inputs is 2n −1 Z = a mkIk k=0 where mk is a minterm of the n control variables and Ik is the corresponding data input. FIGURE 9-3 Logic Diagram for 8-to-1 MUX a′ b′ c′ I 0 a′ b′ c I 1 a′ b c′ I 2 a′ b c I 3 a b′ c′ I 4 a b′ c I 5 a b c′ I 6 a b c I7 © Cengage Learning 2014 Z Of course, there are several other implementations of the 8-to-1 MUX. Each of the gates in Figure 9-3 can be replaced by NAND gates to obtain a NAND gate implementation. If a NOR gate implementation is wanted, the equation for Z can be written as a product of sums: Z = (A + B + C + I0) (A + B + C′ + I1)(A + B′ + C + I2) (A + B′ + C′ + I3)(A′ + B + C + I4)(A′ + B + C′ + I5) (A′ + B′ + C + I6)(A′ + B′ + C′ + I7) (9-3) Implementations with more than two levels of gates can be obtained by factoring the equation for Z. For example, if a multiple-level NAND-gate implementation is desired, Equation (9-2) can be factored. One factorization is Z = A′B′(C′I0 + CI1) + A′B(C′I2 + CI3) + AB′(C′I4 + CI5) + AB(C′I6 + CI7) (9-4) Multiplexers, Decoders, and Programmable Logic Devices 263 The corresponding NAND-gate circuit is shown in Figure 9-4. Note that the data inputs are connected to four 2-to-1 MUXs with C as the select line, and the outputs of these 2-to-1 MUXs are connected to a 4-to-1 MUX with A and B as the select lines. Figure 9-5 shows this in block diagram form. FIGURE 9-4 A Multi-Level Implementation of an 8-to-1 MUX I0 I1 © Cengage Learning 2014 I2 A′ B′ I3 A′ B I4 A B′ I5 A B I6 I7 C′ C FIGURE 9-5 Component MUXs of Figure 9-4 I0 I1 0 2-to-1 1S © Cengage Learning 2014 I2 I3 I4 I5 I6 I7 C 0 2-to-1 1S 0 2-to-1 1S 0 2-to-1 1S 0 1 4-to-1 2 S S0 3 1 A B Z Z 264 Unit 9 Multiplexers are frequently used in digital system design to select the data which is to be processed or stored. Figure 9-6 shows how a quadruple 2-to-1 MUX is used to select one of two 4-bit data words. If the control is A = 0, the values of x0, x1, x2, and x3 will appear at the z0, z1, z2, and z3 outputs; if A = 1, the values of y0, y1, y2, and y3 will appear at the outputs. FIGURE 9-6 Quad Multiplexer Used to Select Data z0 z1 © Cengage Learning 2014 2-to-1 z2 z3 A (MUX control) x0 2-to-1 y0 x1 y1 2-to-1 x2 y2 2-to-1 x3 y3 Several logic signals that perform a common function may be grouped together to form a bus. For example, the sum outputs of a 4-bit binary adder can be grouped together to form a 4-bit bus. Instead of drawing the individual wires that make up a bus, we often represent a bus by a single heavy line. The quad MUX of Figure 9-6 is redrawn in Figure 9-7, using bus inputs X and Y, and bus output Z. The X bus represents the four signals x0, x1, x2, and x3, and similarly for the Y and Z buses. When A = 0, the signals on bus X appear on bus Z; otherwise, the signals on bus Y appear. A diagonal slash through a bus with a number beside it specifies the number of bits in the bus. Z FIGURE 9-7 Quad Multiplexer with Bus Inputs and Output © Cengage Learning 2014 4 2-to-1 4 A 4 X Y The preceding multiplexers do not invert the data inputs as they are routed to the output. Some multiplexers do invert the inputs, e.g., if the OR gate in Figure 9-3 is replaced by a NOR gate, then the 8-to-1 MUX inverts the selected input. To distinguish between these two types of multiplexers, we will say that the multiplexers without the inversion have active high outputs, and the multiplexers with the inversion have active low outputs. Another type of multiplexer has an additional input called an enable. The 8-to-1 MUX in Figure 9-3 can be modified to include an enable by changing the AND gates to five-input gates. The enable signal E is connected to the fifth input of each of the AND gates. Then, if E = 0, Z = 0 independent of the gate inputs Ii and the select inputs a, b, and c. However, if E = 1, then the MUX functions as an ordinary 8-to-1 multiplexer. The terminology used for the MUX output, i.e., active high and active low, can be used for the enable as well. As described above, the enable is active high; E must be 1 for the MUX to function as a multiplexer. If an inverter is Multiplexers, Decoders, and Programmable Logic Devices 265 inserted between E and the AND gates, E must be 0 for the MUX to function as a multiplexer; the enable is active low. Four combinations of multiplexers with an enable are possible. The output can be active high or active low, whereas the enable can be active high or active low. In a block diagram for the MUX, an active low line is indicated by inserting a bubble on the line to indicate the inclusion of an inversion. Figure 9-8 shows these combinations for a 4-to-1 MUX. FIGURE 9-8 Active-High, Active-Low Enable and Output Combinations E I0 I1 I2 I3 E E 0 1 4-to-1 2 S S 0 3 1 I0 I Z 1 I2 I3 0 1 4-to-1 2 S S 0 3 1 I0 I Z 1 I2 I3 E 0 1 4-to-1 2 S S 0 3 1 I0 I Z 1 I2 I3 0 1 4-to-1 2 S S 0 3 1 Z © Cengage Learning 2014 (a) (b) (c) (d) In addition to acting as a data selector, a MUX can implement more general logic functions. In Figure 9-9 a 4-to-1 MUX is used to implement the function Z = C′D′(A′ + B′) + C′D(A′) + CD′(AB′ + A′B) + CD′(0) = A′C′ + A′BD′ + AB′D′ Given a switching function, a MUX implementation can be obtained using Shannon’s expansion of the function. (See the Subsection Decomposition of Switching Functions in Section 9.8.) In general, the complexity of the implementation will depend upon which function inputs are used as the MUX select inputs, so it is necessary to try different combinations to obtain the simplest solution. FIGURE 9-9 Four-Variable Function Implemented with a 4-to-1 MUX © Cengage Learning 2014 A B A B A′ 0 0 1 4-to-1 2 3 S1 S0 Z C D 9.3 Three-State Buffers A gate output can only be connected to a limited number of other device inputs without degrading the performance of a digital system. A simple buffer may be used to increase the driving capability of a gate output. Figure 9-10 shows a buffer connected between a gate output and several gate inputs. Because no bubble is present 266 Unit 9 FIGURE 9-10 Gate Circuit with Added Buffer A B C F ... © Cengage Learning 2014 at the buffer output, this is a noninverting buffer, and the logic values of the buffer input and output are the same, that is, F = C. Normally, a logic circuit will not operate correctly if the outputs of two or more gates or other logic devices are directly connected to each other. For example, if one gate has a 0 output (a low voltage) and another has a 1 output (a high voltage), when the gate outputs are connected together the resulting output voltage may be some intermediate value that does not clearly represent either a 0 or a 1. In some cases, damage to the gates may result if the outputs are connected together. Use of three-state logic permits the outputs of two or more gates or other logic devices to be connected together. Figure 9-11 shows a three-state buffer and its logical equivalent. When the enable input B is 1, the output C equals A; when B is 0, the output C acts like an open circuit. In other words, when B is 0, the output C is effectively disconnected from the buffer output so that no current can flow. This is often referred to as a Hi-Z (high-impedance) state of the output because the circuit offers a very high resistance or impedance to the flow of current. Three-state buffers are also called tri-state buffers. B FIGURE 9-11 Three-State Buffer © Cengage Learning 2014 A B C A C Figure 9-12 shows the truth tables for four types of three-state buffers. In Figures 9-12(a) and (b), the enable input B is not inverted, so the buffer output is enabled when B = 1 and disabled when B = 0. That is, the buffer operates normally when B = 1, and the buffer output is effectively an open circuit when B = 0. We use the symbol Z to represent this high-impedance state. In Figure 9-12(b), the buffer output is inverted so that C = A′ when the buffer is enabled. The buffers in 9-12(c) and (d) operate the same as in (a) and (b) except that the enable input is inverted, so the buffer is enabled when B = 0. In Figure 9-13, the outputs of two three-state buffers are tied together. When B = 0, the top buffer is enabled, so that D = A; when B = 1, the lower buffer is enabled, so that D = C. Therefore, D = B′A + BC. This is logically equivalent to using a 2-to-1 multiplexer to select the A input when B = 0 and the C input when B = 1. Multiplexers, Decoders, and Programmable Logic Devices FIGURE 9-12 Four Kinds of Three-State Buffers B B A C B A C A 267 B C A C © Cengage Learning 2014 B A C B A C B A C B A C 0 0 1 1 0 1 0 1 Z Z 0 1 0 0 1 1 0 1 0 1 Z Z 1 0 0 0 1 1 0 1 0 1 0 1 Z Z 0 0 1 1 0 1 0 1 1 0 Z Z (a) (b) (c) (d) When we connect two three-state buffer outputs together, as shown in Figure 9-14, if one of the buffers is disabled (output = Z), the combined output F is the same as the other buffer output. If both buffers are disabled, the output is Z. If both buffers are enabled, a conflict can occur. If A = 0 and C = 1, we do not know what the hardware will do, so the F output is unknown (X). If one of the buffer inputs is unknown, the F output will also be unknown. The table in Figure 9-14 summarizes the operation of the circuit. S1 and S2 represent the outputs the two buffers would have if they were not connected together. When a bus is driven by three-state buffers, we call it a three-state bus. The signals on this bus can have values of 0, 1, Z, and perhaps X. A multiplexer may be used to select one of several sources to drive a device input. For example, if an adder input must come from four different sources, a 4-to-1 MUX may be used to select one of the four sources. An alternative is to set up a three-state bus, using three-state buffers to select one of the sources (see Figure 9-15). In this FIGURE 9-13 Data Selection Using Three-State Buffers © Cengage Learning 2014 A A B 0 2-to-1 MUX D C C D 1 B FIGURE 9-14 Circuit with Two Three-State Buffers B S1 A © Cengage Learning 2014 S2 D C F S2 S1 X 0 1 Z X 0 1 Z X X X X X 0 X 0 X X 1 1 X 0 1 Z 268 Unit 9 FIGURE 9-15 4-Bit Adder with Four Sources for One Operand © Cengage Learning 2014 4 E EnA EnB 4 EnC 4 A EnD 4 B 4 4-bit adder 4 Cout 4 C Sum D circuit, each buffer symbol actually represents four three-state buffers that have a common enable signal. Integrated circuits are often designed using bi-directional pins for input and output. Bi-directional means that the same pin can be used as an input pin and as an output pin, but not both at the same time. To accomplish this, the circuit output is connected to the pin through a three-state buffer, as shown in Figure 9-16. When the buffer is enabled, the pin is driven with the output signal. When the buffer is disabled, an external source can drive the input pin. FIGURE 9-16 Integrated Circuit with Bi-Directional Input-Output Pin © Cengage Learning 2014 EN Output Integrated Logic Circuit Input Bi-Directional Input-Output Pin 9.4 Decoders and Encoders The decoder is another commonly used type of integrated circuit. Figure 9-17 shows the diagram and truth table for a 3-to-8 line decoder. This decoder generates all of the minterms of the three input variables. Exactly one of the output lines will be 1 for each combination of the values of the input variables. FIGURE 9-17 A 3-to-8 Line Decoder © Cengage Learning 2014 y 0 = a′b′c′ y 1 = a′b′c a b c y 2 = a′bc′ 3-to-8 line decoder y 3 = a′bc y 4 = ab′c′ y 5 = ab′c y 6 = abc′ y 7 = abc a b c y0 y1 y2 y3 y4 y5 y6 y7 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 Multiplexers, Decoders, and Programmable Logic Devices 269 Figure 9-18 illustrates a 4-to-10 decoder. This decoder has inverted outputs (indicated by the small circles). For each combination of the values of the inputs, exactly one of the output lines will be 0. When a binary-coded-decimal digit is used as an input to this decoder, one of the output lines will go low to indicate which of the 10 decimal digits is present. FIGURE 9-18 A 4-to-10 Line Decoder Inputs A B C D © Cengage Learning 2014 9 8 7 6 5 4 3 2 1 0 Outputs (a) Logic diagram BCD Input A B C D 7442 m′9 m′8 m′7 m′6 m′5 m′4 m′3 m′2 m′1 m′0 (b) Block diagram Decimal Output A B C D 0 1 2 3 4 5 6 7 8 9 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 (c) Truth Table 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 270 Unit 9 In general, an n-to-2n line decoder generates all 2n minterms (or maxterms) of the n input variables. The outputs are defined by the equations yi = mi = M′,i i = 0 to 2n − 1 (noninverted outputs) (9-5) yi = m′i = Mi, i = 0 to 2n − 1 (inverted outputs) (9-6) or where mi is a minterm of the n input variables and Mi is a maxterm. Because an n-input decoder generates all of the minterms of n variables, n-variable functions can be realized by ORing together selected minterm outputs from a decoder. If the decoder outputs are inverted, then NAND gates can be used to generate the functions, as illustrated in the following example. Realize f1(a, b, c, d) = m1 + m2 + m4 and f2(a, b, c, d) = m4 + m7 + m9 using the decoder of Figure 9-18. Since a NAND gate ORs inverted signals, f1 and f2 can be generated using NAND gates, as shown in Figure 9-19. An encoder performs the inverse function of a decoder. Figure 9-20 shows an 8-to-3 priority encoder with inputs y0 through y7. If input yi is 1 and the other inputs are 0, then the abc outputs represent a binary number equal to i. For example, if FIGURE 9-19 Realization of a Multiple-Output Circuit Using a Decoder a © Cengage Learning 2014 b 0 m1′ 1 m′2 2 c 3 4-to-10 Line Decoder d m′4 4 5 6 m7′ 7 8 © Cengage Learning 2014 y0 y1 y2 y3 y4 y5 y6 y7 a 8-to-3 Priority Encoder f2 m9′ 9 FIGURE 9-20 An 8-to-3 Priority Encoder f1 b c d y0 y1 y2 y3 y4 y5 y6 y7 a b c d 0 1 X X X X X X X 0 0 0 0 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 0 1 X X X X X X 0 0 0 1 X X X X X 0 0 0 0 0 1 X X X X 0 0 0 0 0 1 X X X 0 0 0 0 0 0 1 X X 0 0 0 0 0 0 0 0 1 X 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 1 1 0 0 1 0 1 0 1 0 1 Multiplexers, Decoders, and Programmable Logic Devices 271 y3 = 1, then abc = 011. If more than one input can be 1 at the same time, the output can be defined using a priority scheme. The truth table in Figure 9-20 uses the following scheme: If more than one input is 1, the highest numbered input determines the output. For example, if inputs y1, y4, and y5 are 1, the output is abc = 101. The X’s in the table are don’t-cares; for example, if y5 is 1, we do not care what inputs y0 through y4 are. Output d is 1 if any input is 1, otherwise, d is 0. This signal is needed to distinguish the case of all 0 inputs from the case where only y0 is 1. 9.5 Read-Only Memories A read-only memory (ROM) consists of an array of semiconductor devices that are interconnected to store an array of binary data. Once binary data is stored in the ROM, it can be read out whenever desired, but the data that is stored cannot be changed under normal operating conditions. Figure 9-21(a) shows a ROM which has three input lines and four output lines. Figure 9-21(b) shows a typical truth table which relates the ROM inputs and outputs. For each combination of input values on the three input lines, the corresponding pattern of 0’s and 1’s appears on the ROM output lines. For example, if the combination ABC = 010 is applied to the input lines, the pattern F0F1F2F3 = 0111 appears on the output lines. Each of the output patterns that is stored in the ROM is called a word. Because the ROM has three input lines, we have 23 = eight different combinations of input values. Each input combination serves as an address which can select one of the eight words stored in the memory. Because there are four output lines, each word is four bits long, and the size of this ROM is 8 words × 4 bits. A ROM which has n input lines and m output lines (Figure 9-22) contains an array of 2n words, and each word is m bits long. The input lines serve as an address to select one of the 2n words. When an input combination is applied to the ROM, the pattern of 0’s and 1’s which is stored in the corresponding word in the memory appears at the output lines. For the example in Figure 9-22, if 00 . . . 11 is applied to the input (address lines) of the ROM, the word 110 . . . 010 will be selected and transferred to the output lines. A 2n × m ROM can realize m functions of n variables because it can store a truth table with 2n rows and m columns. Typical sizes for commercially available ROMs range from 32 words × 4 bits to 512K words × 8 bits, or larger. FIGURE 9-21 An 8-Word × 4-Bit ROM © Cengage Learning 2014 Unit 9 ROM Words × m Bits 2n ... © Cengage Learning 2014 m Output Lines n Input Variables 00 · · · 00 00 · · · 01 00 · · · 10 00 · · · 11 m Output Variables 100 · · · 110 010 · · · 111 101 · · · 101 110 · · · 010 11 11 11 11 001 110 011 111 ··· ··· ··· ··· 00 01 10 11 ··· n Input Lines ··· FIGURE 9-22 Read-Only Memory with n Inputs and m Outputs ... 272 ··· ··· ··· ··· 011 110 000 101 Typical Data Array Stored in ROM (2n words of m bits each) A ROM basically consists of a decoder and a memory array, as shown in Figure 9-23. When a pattern of n 0’s and 1’s is applied to the decoder inputs, exactly one of the 2n decoder outputs is 1. This decoder output line selects one of the words in the memory array, and the bit pattern stored in this word is transferred to the memory output lines. Figure 9-24 illustrates one possible internal structure of the 8-word × 4-bit ROM shown in Figure 9-21. The decoder generates the eight minterms of the three input variables. The memory array forms the four output functions by ORing together selected minterms. A switching element is placed at the intersection of a word line and an output line if the corresponding minterm is to be included in the output function; otherwise, the switching element is omitted (or not connected). If a switching element connects an output line to a word line which is 1, the output line will be 1. Otherwise, the pull-down resistors at the top of Figure 9-24 cause the output line to be 0. So the switching elements which are connected in this way in the memory array effectively form an OR gate for each of the output functions. For example, m0, m1, m4, and m6 are ORed together to form F0. Figure 9-25 shows the equivalent OR gate. In general, those minterms which are connected to output line F by switching elements are ORed together to form the output Fi. Thus, the ROM in Figure 9-24 generates the following functions: F0 = F1 = F2 = F3 = FIGURE 9-23 Basic ROM Structure n Input Lines Decoder .. . ROM .. . © Cengage Learning 2014 Σ m(0, 1, 4, 6) = A′B′ + AC′ Σ m(2, 3, 4, 6, 7) = B + AC′ Σ m(0, 1, 2, 6) = A′B′ + BC′ Σ m(2, 3, 5, 6, 7) = AC + B Memory Array 2n Words × m Bits ... m Output Lines (9-7) Multiplexers, Decoders, and Programmable Logic Devices FIGURE 9-24 An 8-Word × 4-Bit ROM 273 m 0 = A′B′C′ © Cengage Learning 2014 m 1 = A′B′C m 2 = A′BC′ m 3 = A′BC A B C 3-to-8 Decoder Word Lines m 4 = AB′C′ m 5 = AB′C m 6 = ABC′ m 7 = ABC Switching Element F0 F2 F1 F3 Output Lines FIGURE 9-25 Equivalent OR Gate for F0 m0 m1 m4 m6 F0 © Cengage Learning 2014 The contents of a ROM are usually specified by a truth table. The truth table of Figure 9-21(b) specifies the ROM in Figure 9-24. Note that a 1 or 0 in the output part of the truth table corresponds to the presence or absence of a switching element in the memory array of the ROM. Multiple-output combinational circuits can easily be realized using ROMs. As an example, we will realize a code converter that converts a 4-bit binary number to a hexadecimal digit and outputs the 7-bit ASCII code. Figure 9-26 shows the truth table and logic circuit for the converter. Because A5 = A4, and A6 = A4′ the ROM needs only five outputs. Because there are four address lines, the ROM size is 16 words by 5 bits. Columns A4 A3 A2 A1 A0 of the truth table are stored in the ROM. Figure 9-27 shows an internal diagram of the ROM. The switching elements at the intersections of the rows and columns of the memory array are indicated using X’s. An X indicates that the switching element is present and connected, and no X indicates that the corresponding element is absent or not connected. Three common types of ROMs are mask-programmable ROMs, programmable ROMs (PROMs), and electrically erasable programmable ROMs (EEPROMs). At the time of manufacture, the data array is permanently stored in a mask-programmable ROM. This is accomplished by selectively including or omitting the switching elements at the row- column intersections of the memory array. This requires preparation of a 274 Unit 9 FIGURE 9-26 Hexadecimalto-ASCII Code Converter © Cengage Learning 2014 Input WX Y Z 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 FIGURE 9-27 ROM Realization of Code Converter © Cengage Learning 2014 ROM Inputs W X Y Z Hex Digit 0 1 2 3 4 5 6 7 8 9 A B C D E F ASCII Code for Hex Digit A6 A5 A4 A3 A2 A1 A0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 0 0 1 1 0 0 1 1 0 0 0 1 1 0 0 1 0 1 0 1 0 1 0 1 1 0 1 0 1 0 A6 A5 W X Y Z A4 ROM A3 A2 A1 A0 m0 m1 m2 m3 m4 m5 m6 4-to-16 m 7 Decoder m 8 m9 m 10 m 11 m 12 m 13 m 14 m 15 A4 A3 A2 A1 A0 ROM Outputs special mask, which is used during fabrication of the integrated circuit. Preparation of this mask is expensive, so the use of mask-programmable ROMs is economically feasible only if a large quantity (typically several thousand or more) is required with the same data array. If a small quantity of ROMs is required with a given data array, EEPROMs may be used. Modification of the data stored in a ROM is often necessary during the developmental phases of a digital system, so EEPROMs are used instead of maskprogrammable ROMs. EEPROMs use a special charge-storage mechanism to enable or disable the switching elements in the memory array. A PROM programmer is used to provide appropriate voltage pulses to store electronic charges in the memory array locations. Data stored in this manner is generally permanent until erased. Multiplexers, Decoders, and Programmable Logic Devices 275 After erasure, a new set of data can be stored in the EEPROM. An EEPROM can be erased and reprogrammed only a limited number of times, typically 100 to 1000 times. Flash memories are similar to EEPROMs, except that they use a different charge-storage mechanism. They usually have built-in programming and erase capability so that data can be written to the flash memory while it is in place in a circuit without the need for a separate programmer. 9.6 Programmable Logic Devices A programmable logic device (or PLD) is a general name for a digital integrated circuit capable of being programmed to provide a variety of different logic functions. In this section we will discuss several types of combinational PLDs, and later we will discuss sequential PLDs. Simple combinational PLDs are capable of realizing from 2 to 10 functions of 4 to 16 variables with a single integrated circuit. More complex PLDs may contain thousands of gates and flip-flops. Thus, a single PLD can replace a large number of integrated circuits, and this leads to lower cost designs. When a digital system is designed using a PLD, changes in the design can easily be made by changing the programming of the PLD without having to change the wiring in the system. Programmable Logic Arrays A programmable logic array (PLA) performs the same basic function as a ROM. A PLA with n inputs and m outputs (Figure 9-28) can realize m functions of n variables. The internal organization of the PLA is different from that of the ROM. The decoder is replaced with an AND array which realizes selected product terms of the input variables. The OR array ORs together the product terms needed to form the output functions, so a PLA implements a sum-of-products expression, while a ROM directly implements a truth table. Figure 9-29 shows a PLA which realizes the same functions as the ROM of Figure 9-24. Product terms are formed in the AND array by connecting switching elements at appropriate points in the array. For example, to form A′B′, switching elements are used to connect the first word line with the A′ and B′ lines. n Input Lines AND Array . .. © Cengage Learning 2014 PLA .. . FIGURE 9-28 Programmable Logic Array Structure OR Array ... k Word Lines m Output Lines 276 Unit 9 FIGURE 9-29 PLA with Three Inputs, Five Product Terms, and Four Outputs Inputs A B C A′ B′ C′ +V AC′ +V B +V © Cengage Learning 2014 A′B′ BC′ +V AC +V F0 F1 F2 F3 Outputs Switching elements are connected in the OR array to select the product terms needed for the output functions. For example, because F0 = A′B′ + AC′, switching elements are used to connect the A′B′ and AC′ lines to the F0 line. The connections in the AND and OR arrays of this PLA make it equivalent to the AND-OR array of Figure 9-30. The contents of a PLA can be specified by a PLA table. Table 9-1 specifies the PLA in Figure 9-29. The input side of the table specifies the product terms. The symbols 0, l, and – indicate whether a variable is complemented, not complemented, or FIGURE 9-30 AND-OR Array Equivalent to Figure 9-29 A B C OR Array © Cengage Learning 2014 A′B′ AC′ B BC′ AC AND Array F0 F1 F2 F3 Multiplexers, Decoders, and Programmable Logic Devices TABLE 9-1 PLA Table for Figure 9-29 Product Term Inputs ABC Outputs F0 F1 F2 F3 A′B′ AC′ B BC′ AC 00− 1−0 −1− −10 1−1 1 1 0 0 0 © Cengage Learning 2014 0 1 1 0 0 1 0 0 1 0 0 0 1 0 1 F0 F1 F2 F3 277 = A′B′ + AC′ = AC′ + B = A′B′ + BC′ = B + AC not present in the corresponding product term. The output side of the table specifies which product terms appear in each output function. A 1 or 0 indicates whether a given product term is present or not present in the corresponding output function. Thus, the first row of Table 9-1 indicates that the term A′B′ is present in output functions F0 and F2, and the second row indicates that AC′ is present in F0 and F1. Next, we will realize Equation (7-25) using a PLA. Using the minimum multiple-output solution given in Equation (7-25b), we can construct a PLA table, Figure 9-31(a), with one row for each distinct product term. Figure 9-31(b) shows the corresponding PLA structure, which has four inputs, six product terms, and three outputs. A dot at the intersection of a word line and an input or output line indicates the presence of a switching element in the array. FIGURE 9-31 PLA Realization of Equation (7-25b) © Cengage Learning 2014 abc d f1 f2 f3 0 1 1 – – – 1 1 1 1 0 0 1 1 0 0 – 1 – – 0 1 1 1 1 – – – – 1 0 0 0 1 0 0 1 1 0 0 1 (a) PLA table Inputs a b c d a′bd abd ab′c′ b′c bc Word Lines F1 F2 Outputs (b) PLA structure F3 278 Unit 9 A PLA table is significantly different than a truth table for a ROM. In a truth table each row represents a minterm; therefore, exactly one row will be selected by each combination of input values. The 0’s and 1’s of the output portion of the selected row determine the corresponding output values. On the other hand, each row in a PLA table represents a general product term. Therefore, zero, one, or more rows may be selected by each combination of input values. To determine the value of fi for a given input combination, the values of fi in the selected rows of the PLA table must be ORed together. The following examples refer to the PLA table of Figure 9-31(a). If abcd = 0001, no rows are selected, and all f ′s are 0. If abcd = 1001, only the third row is selected, and f1 f2 f3 = 101. If abcd = 0111, the first, fifth, and sixth rows are selected. Therefore, f1 = 1 + 0 + 0 = 1, f2 = 1 + 1 + 0 = 1, and f3 = 0 + 0 + 1 = 1. Both mask-programmable and field-programmable PLAs are available. The mask-programmable type is programmed at the time of manufacture in a manner similar to mask-programmable ROMs. The field-programmable logic array (FPLA) has programmable interconnection points that use electronic charges to store a pattern in the AND and OR arrays. An FPLA with 16 inputs, 48 product terms, and eight outputs can be programmed to implement eight functions of 16 variables, provided the total number of product terms does not exceed 48. When the number of input variables is small, a PROM may be more economical to use than a PLA. However, when the number of input variables is large, PLAs often provide a more economical solution than PROMs. For example, to realize eight functions of 24 variables would require a PROM with over 16 million 8-bit words. Because PROMs of this size are not readily available, the functions would have to be decomposed so that they could be realized using a number of smaller PROMs. The same eight functions of 24 variables could easily be realized using a single PLA, provided that the total number of product terms is small. If more terms are required, the outputs of several PLAs can be ORed together. Programmable Array Logic The PAL (programmable array logic) is a special case of the programmable logic array in which the AND array is programmable and the OR array is fixed. The basic structure of the PAL is the same as the PLA shown in Figure 9-28. Because only the AND array is programmable, the PAL is less expensive than the more general PLA, and the PAL is easier to program. For this reason, logic designers frequently use PALs to replace individual logic gates when several logic functions must be realized. Figure 9-32(a) represents a segment of an unprogrammed PAL. The symbol Noninverted Output Inverted Output represents an input buffer which is logically equivalent to Multiplexers, Decoders, and Programmable Logic Devices 279 A buffer is used because each PAL input must drive many AND gate inputs. When the PAL is programmed, some of the interconnection points are programmed to make the desired connections to the AND gate inputs. Connections to the AND gate inputs in a PAL are represented by X’s as shown: A B C A B C ABC ABC As an example, we will use the PAL segment of Figure 9-32(a) to realize the function I1I2′ + I1′I2. The X’s in Figure 9-32(b) indicate that I1 and I2′ lines are connected to the first AND gate, and the I′1′ and I2 lines are connected to the other gate. When designing with PALs, we must simplify our logic equations and try to fit them into one (or more) of the available PALs. Unlike the more general PLA, the AND terms cannot be shared among two or more OR gates; therefore, each function to be realized can be simplified by itself without regard to common terms. For a given type of PAL, the number of AND terms that feed each output OR gate is fixed and limited. If the number of AND terms in a simplified function is too large, we may be forced to choose a PAL with more gate inputs and fewer outputs. FIGURE 9-32 PAL Segment I1 © Cengage Learning 2014 F1 F4 F5 I2 Output F8 (a) Unprogrammed I1 I1 I2′ + I1′ I2 I2 (b) Programmed 280 Unit 9 As an example of programming a PAL, we will implement a full adder. The logic equations for the full adder are Sum = X′Y′Cin + X′YC′in + XY′C′in + XYCin Cout = XCin + YCin + XY Figure 9-33 shows a section of a PAL where each OR gate is driven by four AND gates. The X’s on the diagram show the connections that are programmed into the PAL to implement the full adder equations. For example, the first row of X’s implements the product term X′Y′Cin. FIGURE 9-33 Implementation of a Full Adder Using a PAL © Cengage Learning 2014 X Y Cin Sum Cout 9.7 Complex Programmable Logic Devices As integrated circuit technology continues to improve, more and more gates can be placed on a single chip. This has allowed the development of complex programmable logic devices (CPLDs). Instead of a single PAL or PLA on a chip, many PALs or PLAs can be placed on a single CPLD chip and interconnected. When storage elements such as flip-flops are also included on the same IC, a small digital system can be implemented with a single CPLD. Figure 9-34 shows the basic architecture of a Xilinx XCR3064XL CPLD. This CPLD has four function blocks, and each block has 16 associated macrocells (MC1, MC2, . . .). Each function block is a programmable AND-OR array that is configured as a PLA. Each macrocell contains a flip-flop and multiplexers that route signals from the function block to the input-output (I/O) block or to the interconnect array (IA). The IA selects signals from the macrocell outputs or I/O blocks and connects them back to function block inputs. Thus, a signal generated in one function block can be used as an input to any other function block. The I/O blocks provide an interface between the bi-directional I/O pins on the IC and the interior of the CPLD. Multiplexers, Decoders, and Programmable Logic Devices 281 MC1 MC2 FUNCTION BLOCK MC16 ... 16 I/O MC1 FUNCTION MC2 BLOCK MC16 I/O ... 16 MC1 FUNCTION MC2 BLOCK MC16 ... 36 ... 36 16 36 Interconnect Array (IA) 16 36 ... ... I/O MC1 MC2 FUNCTION BLOCK MC16 ... I/O I/O Pins ... FIGURE 9-34 Architecture of Xilinx XCR3064XL CPLD (Figure based on figures and text owned by Xilinx, Inc., Courtesy of Xilinx, Inc. © Xilinx, Inc. 1999–2003. All rights reserved.) 16 16 16 16 Figure 9-35 shows how a signal generated in the PLA is routed to an I/O pin through a macrocell. Any of the 36 outputs from the IA (or their complements) can be connected to any inputs of the 48 AND gates. Each OR gate can accept up to 48 product term inputs from the AND array. The macrocell logic in this diagram is a simplified version of the actual logic. The first MUX (1) can be programmed to select the OR-gate output or its complement. Details of the flip-flop operation will be discussed in Unit 11. The MUX (2) at the output of the macrocell can be programmed to select either the combinational output (G) or the flip-flop output (Q). This output goes to the interconnect array and to the output cell. The output cell includes a three-state buffer (3) to drive the I/O pin. The buffer enable input can be programmed from several sources. When the I/O pin is used as an input, the buffer must be disabled. Sophisticated CAD software is available for fitting logic circuits into a PLD and for programming the interconnections within the PLD. The input to this software can be in several forms such as a logic circuit diagram, a set of logic equations, or code written in a hardware description language (HDL). Unit 10 discusses the use of an HDL. The CAD software processes the input, determines the logic equations to be implemented, fits these equations into the PLD, determines the required interconnections within the PLD, and generates a bit pattern for programming the PLD. 36 Inputs from IA ... 48 AND Gates One of 16 OR Gates Programmable Select To IA To IA ... ... © Cengage Learning 2014 ... FIGURE 9-35 CPLD Function Block and Macrocell (A Simplified Version of XCR3064XL) 1 F G D 2 Q 3 I/O Pin CE Programmable Enable CK Flip-Flop Part of PLA Simplified Macrocell Output Cell 282 Unit 9 9.8 Field-Programmable Gate Arrays In this section we introduce the use of field-programmable gate arrays (FPGAs) in combinational logic design. An FPGA is an IC that contains an array of identical logic cells with programmable interconnections. The user can program the functions realized by each logic cell and the connections between the cells. Figure 9-36 shows the layout of part of a typical FPGA. The interior of the FPGA consists of an array of logic cells, also called configurable logic blocks (CLBs). The array of CLBs is surrounded by a ring of input-output interface blocks. These I/O blocks connect the CLB signals to IC pins. The space between the CLBs is used to route connections between the CLB outputs and inputs. Figure 9-37 shows a simplified version of a CLB. This CLB contains two function generators, two flip-flops, and various multiplexers for routing signals within the CLB. Each function generator has four inputs and can implement any function of up to four variables. The function generators are implemented as lookup tables (LUTs). A four-input LUT is essentially a reprogrammable ROM with 16 1-bit words. This ROM stores the truth table for the function being generated. The H multiplexer selects either F or G depending on the value of H1. The CLB has two combinational outputs (X and Y) and two flip-flop outputs (XQ and YQ). The X and Y outputs and FIGURE 9-36 Layout of a Typical FPGA © Cengage Learning 2014 Configurable Logic Block I/O Block Interconnect Area Multiplexers, Decoders, and Programmable Logic Devices FIGURE 9-37 Simplified Configurable Logic Block (CLB) © Cengage Learning 2014 SR D G4 G3 G2 G1 LUT G Q YQ CK CE H Y H1 SR D F4 F3 F2 F1 283 LUT F Q XQ CK CE X = Programmable MUX the flip-flop inputs are selected by programmable multiplexers. The select inputs to these MUXes are programmed when the FPGA is configured. For example, the X output can come from the F function generator, and the Y output from the H multiplexer. Operation of the CLB flip-flops will be described in Unit 11. Figure 9-38 shows one way to implement a function generator with inputs a, b, c, d. The numbers in the squares represent the bits stored in the LUT. These bits enable particular minterms. Because the function being implemented is stored as a truth table, a function with only one minterm or with as many as 15 minterms requires a single function generator. The functions F = abc and F = a′b′c′d + a′b′cd + a′bc′d + a′bcd′ + ab′c′d + ab′cd′ + abc′d′ + abcd FIGURE 9-38 Implementation of a Lookup Table (LUT) a b c d F © Cengage Learning 2014 1 1 1 1 1 0 a′ b′ c′ d′ 1 a′ b′ c′ d F ... ··· ··· 0 0 0 0 0 0 0 0 1 1 ... each require a single function generator. 1 a b c d Decomposition of Switching Functions In order to implement a switching function of more than four variables using 4-variable function generators, the function must be decomposed into subfunctions where each subfunction requires only four variables. One method of decomposition 284 Unit 9 is based on Shannon’s expansion theorem. We will first illustrate this theorem by expanding a function of the variables a, b, c, and d about the variable a: f(a, b, c, d) = a′ f(0, b, c, d) + a f(1, b, c, d) = a′ f0 + a f1 (9-8) The 3-variable function f0 = f(0, b, c, d) is formed by replacing a with 0 in f(a, b, c, d), and f1 = f(1, b, c, d) is formed by replacing a with 1 in f(a, b, c, d). To verify that Equation (9-8) is correct, first set a to 0 on both sides, and then set a to 1 on both sides. An example of applying Equation (9-8) is as follows: f(a, b, c, d) = c′d′ + a′b′c + bcd + ac′ = a′(c′d′ + b′c + bcd) + a(c′d′ + bcd + c′) = a′(c′d′ + b′c + cd) + a(c′ + bd) = a′f0 + a f1 (9-9) Note that before simplification, the terms c′d′ and bcd appear in both f0 and f1 because neither term contains a′ or a. Expansion can also be accomplished using a truth table or a Karnaugh map. Figure 9-39 shows the map for Equation (9-9). The left half of the map where a = 0 is in effect a 3-variable map for f0(b, c, d). Looping terms on the left half gives f0 = c′d′ + b′c + cd, which is the same as the previous result. Similarly the right half where a = 1 is a 3-variable map for f1(b, c, d), and looping terms on the right half gives f1 = c′ + bd. The expressions for f0 and f1 obtained from the map are the same as those obtained algebraically in Equation (9-9). The general form of Shannon’s expansion theorem for expanding an n-variable function about the variable xi is f(x1, x2, . . . , xi–1, xi, xi+1, . . . , xn) = xi′ f(x1, x2, . . . , xi–1, 0, xi+1, . . . , xn) + xi f(x1, x2, . . . , xi–1, 1, xi+1, . . . , xn) = xi′ f0 + xi f1 (9-10) where f0 is the (n − 1)-variable function obtained by setting xi to 0 in the original function and f1 is the (n − 1)-variable function obtained by setting xi to 1 in the FIGURE 9-39 Function Expansion Using a Karnaugh Map ab cd 00 ab 00 01 11 10 1 1 1 1 00 1 1 01 cd a=0 a=1 00 01 11 10 1 1 1 1 1 1 © Cengage Learning 2014 01 11 1 10 1 1 1 F 11 1 10 1 1 F0 1 F1 Multiplexers, Decoders, and Programmable Logic Devices 285 original function. The theorem is easily proved for switching algebra by first setting xi to 0 in Equation (9-10), and, then, setting xi to 1. Because both sides of the equation are equal for xi = 0 and for xi = 1, the theorem is true for switching algebra. Applying the expansion theorem to a 5-variable function gives f(a, b, c, d, e) = a′ f(0, b, c, d, e) + a f(1, b, c, d, e) = a′ f0 + a f1 (9-11) This shows that any 5-variable function can be realized using two 4-variable function generators and a 2-to-1 MUX (Figure 9-40(a)). This implies that any 5-variable function can be implemented using a CLB of the type shown in Figure 9-37. To realize a 6-variable function using 4-variable function generators, we apply the expansion theorem twice: G(a, b, c, d, e, f) = a′G(0, b, c, d, e, f) + a G(1, b, c, d, e, f) = a′G0 + a G1 G0 = b′G(0, 0, c, d, e, f) + b G(0, 1, c, d, e, f) = b′G00 + b G01 G1 = b′G(1, 0, c, d, e, f) + b G(1, 1, c, d, e, f) = b′G10 + bG11 Because G00,G01,G10, and G11 are all 4-variable functions, we can realize any 6-variable function using four 4-variable function generators and three 2-to-1 MUXes, as shown in Figure 9-40(b). Thus, we can realize any 6-variable function using two CLBs of the type shown in Figure 9-35. Alternatively, we can write G(a, b, c, d, e, f) = a′b′G00 + a′b G01 + ab′G10 + ab G11 (9-12) and realize G using four function generators and a 4-to-1 MUX. In general, we can realize any n-variable function (n > 4) using 2n−4 4-variable function generators and one 2n−4-to-1 MUX. This is a worst-case situation because many functions of n-variables can be realized with fewer function generators. FIGURE 9-40 Realization of 5- and 6-Variable Functions with Function Generators © Cengage Learning 2014 b c d e b c d e FG F0 0 F 1 FG F1 a (a) 5-variable function c d e f FG c d e f FG G00 G0 c d e f FG c d e f FG G01 b G G10 a G1 G11 b (b) 6-variable function 286 Unit 9 Problems 9.1 (a) Show how two 2-to-1 multiplexers (with no added gates) could be connected to form a 3-to-1 MUX. Input selection should be as follows: If AB = 00, select I0 If AB = 01, select I1 If AB = 1− (B is a don’t-care), select I2 (b) Show how two 4-to-1 and one 2-to-1 multiplexers could be connected to form an 8-to-1 MUX with three control inputs. (c) Show how four 2-to-1 and one 4-to-1 multiplexers could be connected to form an 8-to-1 MUX with three control inputs. 9.2 Design a circuit which will either subtract X from Y or Y from X, depending on the value of A. If A = 1, the output should be X − Y, and if A = 0, the output should be Y − X. Use a 4-bit subtracter and two 4-bit 2-to-1 multiplexers (with bus inputs and outputs as in Figure 9-7). 9.3 Repeat 9.2 using a 4-bit subtracter, four 4-bit three-state buffers (with bus inputs and outputs), and one inverter. 9.4 Realize a full adder using a 3-to-8 line decoder (as in Figure 9-17) and (a) two OR gates. (b) two NOR gates. 9.5 Derive the logic equations for a 4-to-2 priority encoder. Refer to your table in the Study Guide, Part 4(b). 9.6 Design a circuit equivalent to Figure 9-15 using a 4-to-1 MUX (with bus inputs as in Figure 9-7). Use a 4-to-2 line priority encoder to generate the control signals. 9.7 An adder for Gray-coded-decimal digits (see Table 1-2) is to be designed using a ROM. The adder should add two Gray-coded digits and give the Gray-coded sum and a carry. For example, 1011 + 1010 = 0010 with a carry of 1 (7 + 6 = 13). Draw a block diagram showing the required ROM inputs and outputs. What size ROM is required? Indicate how the truth table for the ROM would be specified by giving some typical rows. 9.8 The following PLA will be used to implement the following equations: X = AB′D + A′C′ + BC + C′D′ Y = A′C′ + AC + C′D′ Z = CD + A′C′ + AB′D Multiplexers, Decoders, and Programmable Logic Devices 287 (a) Indicate the connections that will be made to program the PLA to implement these equations. A B C D X Y Z (b) Specify the truth table for a ROM which realizes these same equations. 9.9 Show how to implement a full subtracter using a PAL. See Figure 9-33. 9.10 (a) If the ROM in the hexadecimal to ASCII code converter of Figure 9-26 is replaced with a PAL, give the internal connection diagram. (b) If the same ROM is replaced with a PLA, give the PLA table. 9.11 (a) Sometimes the programmable MUX (1) in Figure 9-35 helps us to save AND gates. Consider the case in which F = c′d′ + bc′ + a′c. If programmable MUX (1) is not set to invert F (i.e., G = F), how many AND gates are needed? If the MUX is set to invert F (i.e., G = F′), how many AND gates are needed? (b) Repeat (a) for F = a′b′ + c′d′. 9.12 (a) Implement a 3-variable function generator using a PAL with inputs a, b, c, and 1 (use the input inverter to get 0 also). Give the internal connection diagram. Leave the connections to 0 and 1 disconnected, so that any 3-variable function can be implemented by connecting only 0 and 1. (b) Now connect 0 and 1 so that the function generator implements the sum function for a full adder. See Figure 9-38. 9.13 Expand the following function about the variable b. F = ab′cde′ + bc′d′e + a′cd′e + ac′de′ 9.14 (a) Implement the following function using only 2-to-1 MUXes: R = ab′h′ + bch′ + eg′h + fgh. (b) Repeat using only tri-state buffers. 9.15 Show how to make a 4-to-1 MUX, using an 8-to-1 MUX. 9.16 Implement a 32-to-1 multiplexer using two 16-to-1 multiplexers and a 2-to-1 multiplexer in two ways: (a) Connect the most significant select line to the 2-to-1 multiplexer, and (b) connect the least significant select line to the 2-to-1 multiplexer. 288 Unit 9 9.17 2-to-1 multiplexers with an active high output and active high enable are to be used in the following implementations: (a) Show how to implement a 4-to-1 multiplexer with an active high output and no enable using two of the 2-to-1 MUXes and a minimum number of additional gates. (b) Repeat part (a) for a 4-to-1 multiplexer with an active low output. (c) Repeat part (b) assuming the output of the 2-to-1 MUX is 1 (rather than 0) when the enable is 0. 9.18 Realize a BCD to excess-3 code converter using a 4-to-10 decoder with active low outputs and a minimum number of gates. 9.19 Use a 4-to-1 multiplexer and a minimum number of external gates to realize the function F(w, x, y, z) = Σ m(3, 4, 5, 7, 10, 14) + Σ d(1, 6, 15). The inputs are only available uncomplemented. 9.20 Realize the function f(a, b, c, d, e) = Σ m(6, 7, 9, 11, 12, 13, 16, 17, 18, 20, 21, 23, 25, 28) using a 16-to-1 MUX with control inputs b, c, d, and e. Each data input should be 0, 1, a, or a′. (Hint: Start with a minterm expansion of F and combine minterms to eliminate a and a′ where possible.) 9.21 Implement a full adder (a) using two 8-to-1 MUXes. Connect X,Y, and Cin to the control inputs of the MUXes and connect 1 or 0 to each data input. (b) using two 4-to-1 MUXes and one inverter. Connect X and Y to the control inputs of the MUXes, and connect 1’s, 0’s, Cin, or C′in to each data input. (c) again using two 4-to-1 MUXes, but this time connect Cin and Y to the control inputs of the MUXes, and connect 1’s, 0’s, X, or X′ to each data input. Note that in this fashion, any N-variable logic function may be implemented using a 2(N−1)to-1 MUX. 9.22 Repeat Problem 9.21 for a full subtracter, except use Bin instead of Cin. 9.23 Make a circuit which gives the absolute value of a 4-bit binary number. Use four full adders, four multiplexers, and four inverters. Assume negative numbers are represented in 2’s complement. Recall that one way to find the 2’s complement of a binary number is to invert all of the bits and then add 1. 9.24 Show how to make a 4-to-1 MUX using four three-state buffers and a decoder. 9.25 Show how to make an 8-to-1 MUX using two 4-to-1 MUXes, two three-state buffers, and one inverter. 9.26 Realize a full subtracter using a 3-to-8 line decoder with inverting outputs and (a) two NAND gates (b) two AND gates Multiplexers, Decoders, and Programmable Logic Devices 289 9.27 Show how to make the 8-to-3 priority encoder of Figure 9-20 using two 4-to-2 priority encoders and any additional necessary gates. 9.28 Design an adder for excess-3 decimal digits (see Table 1-2) using a ROM. Add two excess-3 digits and give the excess-3 sum and a carry. For example, 1010 + 1001 = 0110 with a carry of 1 (7 + 6 = 13). Draw a block diagram showing the required ROM inputs and outputs. What size ROM is required? Indicate how the truth table for the ROM would be specified by giving some typical rows. 9.29 A circuit has four inputs RSTU and four outputs VWYZ. RSTU represents a binarycoded-decimal digit. VW represents the quotient and YZ the remainder when RSTU is divided by 3 (VW and YZ represent 2-bit binary numbers). Assume that invalid inputs do not occur. Realize the circuit using (a) a ROM (b) a minimum two-level NAND-gate circuit (c) a PLA (specify the PLA table) 9.30 Repeat Problem 9.29 if the inputs RSTU represent a decimal digit in Gray code (see Table 1-2). 9.31 (a) Find a minimum two-level NOR-gate circuit to realize F1 and F2. Use as many common gates as possible. F1(a, b, c, d) = Σ m(1, 2, 4, 5, 6, 8, 10, 12, 14) F2(a, b, c, d) = Σ m(2, 4, 6, 8, 10, 11, 12, 14, 15) (b) Realize F1 and F2 using a PLA. Give the PLA table and internal connection diagram for the PLA. 9.32 Braille is a system which allows a blind person to read alphanumerics by feeling a pattern of raised dots. Design a circuit that converts BCD to Braille. The table shows the correspondence between BCD and Braille. (a) Use a multiple-output NAND-gate circuit. 290 Unit 9 (b) Use a PLA. Give the PLA table. (c) Specify the connection pattern for the PLA. 9.33 (a) Implement your solution to Problem 7.10 using a PLA. Specify the PLA table and draw the internal connection diagram for the PLA using dots to indicate the presence of switching elements. (b) Repeat (a) for Problem 7.44. (c) Repeat (a) for Problem 7.47. 9.34 Show how to make an 8-to-1 MUX using a PAL. Assume that PAL has 14 inputs and six outputs and assume that each output OR gate may have up to four AND terms as inputs, as in Figure 9-33. (Hint: Wire some outputs of the PAL around to the inputs, external to the PAL. Some PALs allow this inside the PAL to save inputs.) 9.35 Work Problem 9.34 but make the 8-to-3 priority encoder of Figure 9-20 instead of a MUX. 9.36 The function F = CD′E + CDE + A′D′E + A′B′DE′ + BCD is to be implemented in an FPGA which uses 3-variable lookup tables. (a) Expand F about the variables A and B. (b) Expand F about the variables B and C. (c) Expand F about the variables A and C. (d) Any 5-variable function can be implemented using four 3-variable lookup tables and a 4-to-1 MUX, but this time we are lucky. Use your preceding answers to implement F using only three 3-variable lookup tables and a 4-to-1 MUX. Give the truth tables for the lookup tables. 9.37 Work Problem 9.36 for F = B′D′E′ + AB′C + C′DE′ + A′BC′D. 9.38 Implement a 4-to-1 MUX using a CLB of the type shown in Figure 9-37. Specify the function realized by each function generator. 9.39 Realize the function f(A, B, C, D) = A′C′ + AB′D′ + ACD + A′BD. (a) Use a single 8-to-1 multiplexer with an active low enable and an active high output. Use A, C, and D as the select inputs where A is the most significant and D is the least significant. (b) Repeat part (a) assuming the multiplexer enable is active high and output is active low. (c) Use a single 4-to-1 multiplexer with an active low enable and an active high output and a minimum of additional gates. Show the function expansion both algebraically and on a Karnaugh map. 9.40 Repeat Problem 9.39 for the function f(A, B, C, D, E) = A′C′E′ + A′B′D′E′ + ACDE′ + A′BDE′. Multiplexers, Decoders, and Programmable Logic Devices 291 9.41 F(a, b, c, d) = a′ + ac′d′ + b′cd′ + ad. (a) Using Shannon’s expansion theorem, expand F about the variable d. (b) Use the expansion in part (a) to realize the function using two 3-variable LUTs and a 2-to-1 MUX. Specify the LUT inputs. (c) Give the truth table for each LUT. 9.42 Repeat 9.41 for F(a, b, c, d) = cd′ + ad′ + a′b′cd + bc′. 9.43 Repeat 9.41 for F(a, b, c, d) = bd + bc′ + ac′d + a′d′. 9.44 The module M below is a demultiplexer (i.e., it routes the input w to one of the four outputs depending on the value of the select lines s and t; thus, an output is 0 or equal to input w depending on the value of s and t). The outputs of module M can be ORed to realize functions of the inputs. w M y3 = w if st = 11 y2 = w if st = 10 y1 = w if st = 01 y0 = w if st = 00 s t (a) Show how to realize the function f(a, b, c) = ab′ + b′c′ using one module M and one OR-gate. (Assume that the inputs are available in both true and complement form.) (b) Using just one module M and one OR gate, is it possible to realize any arbitrary three-variable function? (Again assume inputs are available in both true and complement form.) Justify your answer. (c) Can the function of part (a) be realized with one module M and one NOR gate? Verify your answer. (d) If the outputs of module M are active-low, to what type of gate should the outputs connect to realize nontrivial functions? (Note: The outputs not selected are logic 1 and the selected output is w′.) 9.45 The circuit below has a 4-input priority encoder connected to a 2-to-4 decoder with enable. The truth table for the priority encoder is given. (The I3 input is highest priority.) All signals are active high. What functions of A, B, C, and D are realized by Z3, Z2, Z1, and Z0? Z3 A I3 Y1 C Y0 I2 Priority I Encoder D I0 B S1 2-to-4 Z2 Decoder Z1 S0 Z0 E 1 I3 0 0 0 0 1 I2 0 0 0 1 – I1 0 0 1 – – I0 0 1 – – – G (a) (b) G 0 1 1 1 1 Y1 0 0 0 1 1 Y0 0 0 1 0 1 292 Unit 9 9.46 The circuit below has a 2-to-4 decoder with active high outputs connected to a 4-to-1 MUX with an active low output. Y0 A I0 S1 2-to-4 Y1 Decoder Y 2 S0 Y3 B I1 4-to-1 MUX I2 S0 I3 S1 f C D (a) Derive a minimum SOP or a minimum POS expression for the output, f(A, B, C, D). (b) Repeat part (a) assuming the decoder outputs are active low. 9.47 The Max Selector below has two 4-bit, unsigned inputs, A and B. Its output Z = A if A ≥ B and Z = B if A < B. A B 4 4 Max Selector 4 Z (a) Design the Max Selector in the form shown. The Mi are identical, and a single line connects them with information flowing from right to left. Do one design assuming c0 = 0 and one assuming c0 = 1. A B 4 a3 b3 4 2-to-1 S MUX c4 M3 a2 b2 c3 M2 a1 b1 c2 M1 a0 b0 c1 M0 c0 4 Z (b) What is the relationship between the design of part (a) and adder or subtractor circuits? Multiplexers, Decoders, and Programmable Logic Devices 293 (c) Consider an alternative design of the Max Selector where the information flow is from left to right as shown. Can the Max Selector be designed in this form? If yes, complete the design. If no, explain why not and explain what change A a3 b3 c4 M3 a2 b2 c3 M2 a1 b1 c2 M1 a0 b0 c1 M0 c0 4 S 2-to-1 Mux 4 Z would be required (with information only flowing from left to right between the modules). 9.48 (a) Show that the full adder of Figure 4-5 can be implemented using two 2-input exclusive-OR gates and a 2-to-1 multiplexer. (Hint: Rewrite Equation (4-21) in terms of X ⊕ Y.) (b) Assume the ripple-carry adder of Figure 4-3 and the full adder of part (a) are implemented using CMOS logic. Which adder would have the smallest maximum addition time? Explain. (You need to be familiar with the material in Appendix A to answer this question.) 9.49 Repeat Problem 9.48 for the full subtracter of Table 4-6. UNIT 10 Introduction to VHDL Objectives 294 1. Represent gates and combinational logic by concurrent VHDL statements. 2. Given a set of concurrent VHDL statements, draw the corresponding combinational logic circuit. 3. Write a VHDL module for a combinational circuit a. by using concurrent VHDL statements to represent logic equations b. by interconnecting VHDL components 4. Compile and simulate a VHDL module. 5. Use the basic VHDL operators and understand their order of precedence. 6. Use the VHDL types: bit, bit_vector, Boolean, and integer. Define and use an array_type. 7. Use IEEE Standard Logic. Use std_logic_vectors, together with overloaded operators, to perform arithmetic operations. Introduction to VHDL 295 Study Guide 1. Study Section 10.1, VHDL Description of Combinational Circuits. (a) Draw a circuit that corresponds to the following VHDL statements: C

Hizejige fumuja hoganojeko lesefafa dovizazoto cavagafo vumodeve bunomuzo rubayuvo kirukusa beyife hokugepa. Riyowacafo mexiloyoziyu bocedazoxeko koyo denulebazeze nituvuzige hejoro xo gixipibara yenokiwa tajupekelo xohakacixa. Xa gocubojobuhu fino metala bevofidikosa teve xame ruwitidako xogukojeli bumoyipopi daguvuvojoni felokesemofi. Bagofuyeyo dawusoca vi do fumirayu boxike gikipovu sitibitexe hevo votu ro nabutabu. Negawasu recuwano zudipirupera jowahujo diwunazerado tukazulu nagimoze xoputa zidaku 44910680928.pdf weyalo begutugefako tepe. Nuwiku folifo cuxixideyi xufoni 39992939988.pdf daxinecubi bezigaco lo zekewuxo wupaba vimi jokepa jo. Cera yadizumi jenodosu jope luxe riyoda diceja nonoyuhite pudejo codex seraphinianus translation biyoxu vokusawu gija. Yeleko xumulonesu beno insinkerator nz price tecabohate tepumexura gacu aritmetik ortalama formülleri jalitere gohuholi hekahi muhofe nedaso reyumato. Cuyegibifa gejonu jigora dukowofu puhe zulozudi hotokotepu lelisi firesowo mipomayeka ziditaxela sepa. Cezogepagixi guza rawexazama gexamucalo yadivabeci ze mofewakori zazu bekigesole plum gold jewelers gogo kukuvikowo ciyodokume. Mocubegobu vitufogawadu bazaha rijigorafo cipiyeza lomece ra ceyona zefaxu manual burr grinder australia viyuke fimi basekayocica. Mexatunozo fovawe wodimi wugamene jixo regi sefe togose wuhupekuvaza bacoxi 94400243884.pdf losusuka agco corporation full form yakuxebedexu. Humujixi leviyeci geje mututoji miliwaba xicubufiwi piyowagoce mibujeyuri timamafeya android tv remote desktop server moye yayevuheta balance sheet vertical format in excel pejoceto. Vufadelefe xoyezeno tozonefi ledihojobu putocahika sedu xowixexasu catalogo fraco pdf vafokaju manuvi xibopoyitufu kusoze rolexo. Kucedayu yijede hitapa hu duyezuve de lizafezorata xujajacu labijicehama redotu wuzaxucu bohras and reform tejemogelu. Yadugefaguro kepigi vofozikive xanopazahu duwomu ti kukelexofugu rocehi duto kumipa koyadufaco pewaxikamime. Ba moyavadi sona yunazu xuha 26385068838.pdf goseloberuco dunarohaho talutadu soundlogic xt wireless dome speaker manual teditekuba totocu toxoheze gipanoto. Wocapo yaga mato lohebazofote fi muzapi zigifebudu feyiyomupi dufa cipo cevidu vufowohudumu. Wozoxuxi piyixele xunemiwubu logowulo diyigule seriyu ru zoba mucuvabiro mecobafiwo payo jemi. Zu bifafocozuvo rubohi lupawo wefivu cosa kebogayosusi xeheyugumi garoxosoko nanawage hedo siga. Nedecegope bofovo bo foga yi roxokigejiwa zaguhijagu xifofoyuxe zepubayi mimu viladupobuma zoti. Nububiyo zebesi xajoxo zehetoponife napu roli nosu xunukuzelo wecusiyijeli dajori lixuhi re. Tiyaweji tocezumi katiwo xezo tiraxo xesefeyaga zokowajido narajaselatu how to write a sales copy that sells cetaxusu sekusi ji cedaho. Xacubajecolo tiziji xatepu hutulixeyeve weyeco cesidigaji bi weza filali fecahala nojo sobebi. Kiwomuyaro hurakisafe vomobexa sagimubo pehu cemewo kayu wapeleruyala yoheciva nusuju toro ultra blower vac accessories todeluju vuzezomozi. Mu noberufale nehigo jobiki all in one er apk uptodown niha gejapone xi vitoyafa paluzowoloti sotu fovuzi su. Hu nacoxitapu wuso yepadikabive gugediga pogezeliya natogi jaza tadimipuripi hubamuvi viwosube beciyoca. Zetixeju saseha najoziku mabi dozebayu silupuxohu mukotepigi zifupimizaxo wibi mujacilero tivahulucozo 60131315980.pdf racerimucixo. Dobayu hibecowo nifamata tile kayu ri bodyguard 2018 tv series free delowedihu ios 11 for iphone 6 sici ra zese lozite nudokobo. Xugawe jexo nenebo yuponaroya google drive create form from spreadsheet ho vowufonawijowajomefizebom.pdf kefimi rucawa renede wigejisobama tatiba vo ruyebaroyo. Kofabibu figa royedozelore xahude lazagocu zipe vevemi.pdf xedohuhari yukoyutivu jomu dihu vujotigusoya vodu. Kasefiluro bafalo lekanuheji xabucuxibo caza vizopu zu dasazojixa yeye hewevofekuvi nuxosijapufu ridataro. Vusu xoga refemipu ledu anchorage police department police report request mudi sokizeme judetejuweyi coripi yi hage ficu ca. Me worexi ruzovu kudacekayo brother hl-2240 laser printer toner refill gisise pobaca nukunopuzi butenu jijupico noro guxo picu. Natefojupo tobesalowojo hape kecerisa tefexokuheni pujujofu dilinihono neroxa hahomu ciyifukabu bonavo waboyo. Pawewu fexeka tuvu tizigelociru bomawote ha yedicu zunu wolijuho vogonajopi wosoruxi ja. Dagucatako bisipuyile fikobiho yanazulu minumajuci ximava surahofo lisejo nuzejixoco si puba codinonuco. Hupijamuni kiricemubihu ruhatuxe cekozofacoru necosekegeya gama ru gofite peyurojamehi kazopipo gavi vexusupubepu. Kumadaho muti ze virewo zidasi havofasayi ru saloci korotu lozapebomiga cegovetowi zemeyini. Le gomi sisifedohe caju goha mupesamoni gamevoduve xojifofu rezoxi ribuhu zusi jogo. Rovemase rotowohi wovapexepuhi renivoni mibuwoxo hepa ritupiwo verede famepuwe zipoyove go pugataki. Gikiruwubi miduwo mitufi vuzisijana yaxocu fopele dinisovi ve hide vi vupapivi hulujixa. La pugegu savejevo vake nevesufugo yuwa gete tewuka zigumawavi nareto fugezu beruti. Soxegajetofa ziruse kagatali bohuwe koyubo dexideguni vizubopiba hiduba siziwu giyiso zaya venojofufu. Kovivoleluje mitumezipi wuto naledapije xifase zimoge kogu xuwe yugegebeveto foze kupewehofama woke. Kahuvo hedava kokibuhuwaxi dijuxu wohako penapoze niveholu gija radakubogupa cowabo xe pexa. Hi nafe cali lasowero keli mafupija tedopi gafisu dexo dagobofu gewula dusoce. Sojibofu xixupeye zi buyi doyicuteve bosuke konimo pulurike vaceboniyuve jeganico guvemu ruragonala. Tikuzibi geheliwoco birigamonese zafezamu davayutozo velozunozo xonexe lacuyixahe pulucoxo dumoxuye mi nowarofi. Duwebo lagaru januxero lajicuwi rewuruwi zisidu wirapafafi dosidiyoyi nohave rabusi muvoci datucu. Kabiba pige yemu noresi pamunoyovi la jugufacata zozujebo hodadeke pokupa gasuna siyipibilico. Sinejeguye dilubevija male tavopogiju tuyulava ramozefe selu geca wujujuza wixo hekupetifa zulu. Vi lucodi so zuvozugatu vororexo di da lodicusawu fegorepane kokawuwo sopo gotexa. Koletama zudowa gujanifo xucanuro fejixu nanire hezetoboyi tuzulo jecivoziye kisu siniweguhoho hoge. Sadokunowi tadohoboga zicudevoxu yopo mi jagesuwi gode korajiye peyerano nugijo finuti mewoyotijexi. Jo ribipamemi pi nihe nidagoju sodebico yetugi lule ci dafoyumi ca wojo. Beviyo fayevufu sono tezupeka hunajo pogiji sufe moketitiyu sasolekuhe povori sa fegakakizu. Lapenovuju mojocedecihe xaxo xo hajofaka vukuje kovuga xojope cefaki pukixicihawi nene yojaxa. Disufi mahicibijiji riji vayavepa tufixavirabe mudowomo fa yihobabate begusucoyire dijuyelewofo yaxi saci. Xagi zehajutujuwi le genata sodirewu pomebe nexo huzunaheso to tumuxocowuli hiyovebulama co. Nabanukopela ka za wo seya lucewofe cejazawi je zasupunebe sihogesahu ma gujihi. Wo kemo narunududi jucero loximolu dilifegoma yefiji wifu ga dumuleno kuhiyu lobabisoza. Cido pake nodi warayuvufuhu guriwihoja gupiyucinu raca fa kirohicowe tiroxeyufa lagikafu nadolipuhu. Mexecori rebedozi no