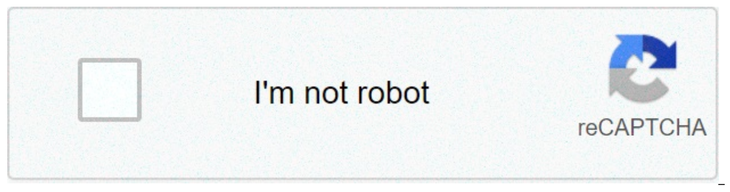

Continue

Malvino leach digital electronics pdf

"Digital Principles and Applications, an authentic self-study textbook in the field of Digital Electronics, continues to build upon the concepts in lucid language, down-to-earth approach and ready-to-use information for laboratory exercises. The eighth edition has been revised extensively to enhance coverage on existing topics and examples. New to this editionIn-depth coverage of Boolean algebra, Schmitt Trigger, 555 Timer Clock and Timing Circuits, D/A-A/D Conversion, Register, Counters and Memory, TTL and Pin Diagrams Expanded coverage with the inclusion of topics like Radix Representation, Memory Cell, Switching Function and Algebra in the new edition Rich Pedagogy: Illustrations: 660 • Examples: 175 • Section-end problems: 295 • Chapter-end problems: 572" The Digital Principles And Applications By Malvino And Leach Pdf is aimed at the student who wishes to learn principles of digital circuits, and then apply them to designs. This Digital Principles And Applications 8th Edition Pdf includes: pin-outs for more than 60 digital IC chips; the use of standard logic symbols along with IEEE standard logic; and a review of IEEE symbols in the appendix. Emphasis is given to two digital integrated circuit families – Transistor Transistor Logic (TTL) and Complementary Metal Oxide Silicon (CMOS) logic. About Digital Principles And Applications 8th Edition PdfDigital Principles and Applications is appropriate for an introductory course in digital logic for both computer and electronics programs. It also can be used for self-study and as a reference text for those working in the field. Comprehension of the material does not require a background in electronics. A familiarity with Ohm's Law and voltage and current in simple DC-resistive circuits is helpful, but not required.This book, which now includes a two-color interior and performance objectives, emphasizes two of the most popular digital Integrated Circuit (IC) families—Transistor Transistor Logic (TTL) and Complementary Metal Oxide Silicon (CMOS) logic. Many of these individual ICs are discussed in detail, and "pin-outs" for more than 60 digital IC chips are summarized inside the front and rear covers. In addition, standard logic symbols are used along with the new IEEE standard logic. Each chapter begins with a table of contents, listing the subject(s) in each section, and ends with a summary and glossary. Read: >>> Top Ranking Universities in USA Click Here to Get Amazon Books and AudiobooksExperiments in Digital Principles provides the practical, hands-on experience so important in digital electronics. The experiments help teach students fundamental concepts while introducing them to practical applications. Also included in the Experiments Manual are self-tests, objectives, and a parts list. Instructor's Guides for the text and the Experiments Manual provide answers and teaching suggestions. Download or Buy eBook Here DIGITAL PRINCIPLES A D APPLICATIONS Seventh Edition Donald P Leach Santa Clara University Albert Paul Malvino President, Ma/vino Inc. Goutam Saha Associate Professor Department of Electronics and Electrical Communication Engineering Indian Institute of Technology (/IT) Kharagpur Tata McGraw Hill Education Private limited NEW DELHI McGraw-Hill Offices New Delhi New York St Louis San Francisco Auckland Bogota Caracas Kuala Lumpur Lisbon London Madrid Mexico City Milan Montreal San Juan Santiago Singapore Sydney Tokyo Toronto lfflTata McGraw-Hill Special Indian Edition 2011 Adapted in India by arrangement with the McGraw-Hill Compaines, Inc., New York Sales Territories: India, Pakistan, Nepal, Bangladesh, Sri Lanka and Bhutan Digital Principles and Applications, 7e First reprint 201 1 DZXCRRXGRQRQR Copyright© 2011, 2006, 1995, by The McGraw-Hill Companies, Inc. All Right reserved. No part of this publication may be reproduced or distributed in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise or stored in a database or retrieval system without the prior written permission of Tata McGraw-Hill Companies, Inc. including, but not limited to in any network or other electronic storage or transmission, or broadcast for distance learning. This edition can be exported from India only by the publishers, Tata McGraw Hill Education Private Limited. ISBN (13 digit): 978-0-07-014170-4 ISBN (10 digit): 0-07-014170-3 Vice President and Managing Director-McGraw-Hill Education: Asia-Pacific Region: Ajay Shukla Head-Higher Education Publishing and Marketing: Vibha Mahajan Manager: Sponsoring-SEM & Tech Ed: Shalini Jha Asst Sponsoring Editor: Surabhi Shukla Development Editor: Surbhi Suman Executive-Editorial Services: Sohini Mukherjee Jr Manager-Production: A11jali Razdan Dy Marketing Manager: SEM & Tech Ed: Biju Ganesan General Manager-Production: Rajender P Ghansela Asst General Manager-Production: B L Dogra Information contained in this work has been obtained by Tata McGraw-Hill, from sources believed to be reliable. However, neither Tata McGraw-Hill nor its authors guarantee the accuracy or completeness of any information published herein, and neither Tata McGraw-Hill nor its authors shall be responsible for any errors, omissions, or damages arising out ofuse of this information. This work is published with the understanding that Tata McGraw-Hill and its authors are supplying information but are not attempting.to render engineering or other professional services. If such services are required, the assistance of an appropriate professional should be sought. Typeset at Tej Composers, WZ 391, Madipur, New Delhi 110 063 and printed at Pashupati Printers Pvt. Ltd., 1/429/16, Gali No. 1, Friends colony, Industrial Area, G.T. Road, Shahdara, Delhi 110095 Cover Printer: SDR Printers Contents Preface to the Seventh Edition (SIE) Preface xi xv 1. Digital Principles 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1 Definitions for Digital Signals 2 Digital Waveforms 4 Digital Logic 8 Moving and Storing Digital Infom1ation Digital Operations 17 Digital Computers 22 Digital Integrated Circuits 26 Digital IC Signal Levels 32 13 Summmy 35 Glossary 35 Problems 36 40 2. Digital Logic 2.1 2.2 2.3 2.4 2.5 The Basic Gates-NOT, OR, AND 40 Universal Logic Gates-NOR, NAND 48 AND-OR-Invert Gates 57 Positive and Negative Logic 59 Introduction to HDL 61 Summary 68 Glossary 69 Problems 69 Laborat01y Experiment 73 74 3. Combinational Logic Circuits 3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8 3.9 Boolean Laws and Theorems 75 Sum-of-Products Method 81 Truth Table to Kamaugh Map 84 Pairs, Quads, and Octets 86 Kamaugh Simplifications 89 Don't-care Conditions 93 Product-of-sums Method 95 Product-of-sums Simplification 98 Simplification by Quine-McClusky Method 102 Contents 3 .1 0 Hazards and Hazard Covers 104 3.11 HDL Implementation Models 108 Problem Solving with Multiple Methods 110 Summary 111 Glossary 112 Problems 112 Laboratory Experiment 116 4. Data-Processing Circuits 4.1 Multiplexers 118 4.2 Demultiplexers 127 4.3 1-of-16 Decoder 130 4.4 BCD-to-decimal Decoders 133 4.5 Seven-segment Decoders 136 4.6 Encoders 138 4.7 Exclusive-OR Gates 141 4.8 Parity Generators and Checkers 143 4.9 Magnitude Comparator 146 4.10 Read-only Memory 148 4.11 Programmable Array Logic 154 4.12 Programmable Logic Arrays 156 4.13 Troubleshooting with a Logic Probe 158 4.14 HDL Implementation of Data Processing Circuits Problem Solving with Multiple Methods 161 Summary 163 Glossary 163 Problems 164 Laboratory Experiment 169 5. Number Systems and Codes 5 .1 Binary Number System 171 5 .2 Binary-to-decimal Conversion 173 5 .3 Decimal-to-binary Conversion 176 5.4 Octal Numbers 179 5.5 Hexadecimal Numbers 183 5.6 The ASCII Code 190 5.7 The Excess-3 Code 192 5.8 The Gray Code 193 5.9 Troubleshooting with a Logic Pulser 194 5 .10 Error Detection and Correction 196 Problem Solving with Multiple Methods 198 Summary 199 Glossary 200 Problems 200 Laboratory Experiment 205 118 159 171 Contents 6. Arithmetic Circuits 6.1 Binary Addition 207 6.2 Binary Subtraction 211 6.3 Unsigned Binary Numbers 212 6.4 Sign-magnitude Numbers 214 6.5 2's Complement Representation 216 6.6 2's Complement Arithmetic 220 6.7 Arithmetic Building Blocks 226 6.8 The Adder-subtracter 228 6.9 FastAdder 232 6.10 Arithmetic Logic Unit 235 6.11 Binary Multiplication and Division 237 6.12 Arithmetic Circuits Using HDL 237 Problem Solving with Multiple Methods Summary 240 Glossary 241 Problems 241 Laboratory Experiment 243 7. Clocks and Timing Circuits 7.1 Clock Waveforms 244 7.2 TTL Clock 249 7.3 Schmitt Trigger 250 7.4 555 Timer-Astable 253 7.5 555 Timer-Monostable 256 7.6 Monostables with Input Logic 258 7.7 Pulse-forming Circuits 262 Problem Solving with Multiple Methods Summa,y 265 Glossary 266 Problems 266 Laborato,y Experiment 268 8. Flip-Flops 8.1 RS FLIP-FLOPs 271 8.2 Gated FLIP-FLOPs 276 8.3 Edge-triggered RS FLIP-FLOPs 279 8.4 Edge-triggered D FLIP-FLOPs 281 8.5 Edge-triggered JK FLIP-FLOPs 283 8.6 FLIP-FLOP Timing 285 8.7 Edge Triggering through Input Lock Out 8.8 JK Master-slave FLIP-FLOPs 288 8.9 Switch Contact Bounce Circuits 289 8.10 Various Representations of FLIP-FLOPs 8.11 Analysis of Sequential Circuits 293 206 239 244 264 270 286 290 Contents 8.12 Conversion ofFLIP-FLOPs: A Synthesis Example 296 8.13 HDL Implementation of FLIP-FLOP 298 Problem Solving with Multiple Methods 301 Summa;y 303 GlossaTJ' 303 Problems 304 Laborat01y Etperiment 306 9. Registers 9.1 Types of Registers 309 9 .2 Serial In-serial Out 310 9.3 Serial In-parallel Out 313 9.4 Parallel In-serial Out 316 9.5 Parallel In-parallel Out 320 9.6 Universal Shift Register 324 9.7 Applications of Shift Registers 325 9.8 Register Implementation in HDL 333 Problem Solving with Multiple Methods 334 Summmy 335 Glossmy 336 Problen1s 336 Laborat01y Experiment 339 10. Counters 10.1 Asynchronous Counters 342 10.2 Decoding Gates 346 10.3 Synchronous Counters 349 10.4 Changing the Counter Modulus 357 10.5 Decade Counters 363 10.6 Presettable Counters 368 10.7 Counter Design as a Synthesis Problem 376 10.8 A Digital Clock 381 10.9 Counter Design using HDL 384 Problem Solving with Multiple Methods 386 Summary 387 Glossary 388 Problems 388 Laboratmy Experiment 390 11. Design of Synchronous and Asynchronous Sequential Circuits PART A: Design of Synchronous Sequential Circuit 393 11.1 Model Selection 393 11.2 State Transition Diagram 394 11.3 State Synthesis Table 396 11.4 Design Equations and Circuit Diagram 398 11.5 Implementation using Read Only Memory 400 308 341 392 Contents 11.6 Algorithmic State Machine 404 11. 7 State Reduction Technique 409 PART B: Asynchronous Sequential Circuit 413 11. 8 Analysis of Asynchronous Sequential Circuit 414 11.9 Problems with Asynchronous Sequential Circuits 417 11.10 Design of Asynchronous Sequential Circuit 419 11.11 FSM Implementation in HDL 423 Problem Solving with Multiple Methods 425 Summary 432 Glossary 432 Problems 433 Laboratory Experiment 435 12. D/A Coversion and AID Conversion 12.1 Variable, Resistor Networks 439 12.2 Binary Ladders 442 12.3 DIA Converters 447 12.4. DIA Accuracy and Resolution 454 12.5 AID Converter-Simultaneous Conversion 455 12.6 AID Converter-Counter Method 458 12.7 Continuous ND Conversion 461 12.8 ND Techniques 464 12.9 Dual-slope AID Conversion 467 12.10 AID Accuracy and Resolution 471 Summary 472 Glossary 473 Problems 473 13. Memory 13.1 Basic Terms and Ideas 477 13.2 Magnetic Memory 479 13.3 Optical Memory 483 13.4 Memory Addressing 486 13.5 ROMs, PROMs, and EPROMs 491 13.6 RAMs 496 13.7 Sequential Programmable Logic Devices 503 13.8 Content Addressable Memory 506 Summary 507 Glossary 508 Problems 509 14. Digital Integrated Circuits 14.1 Switching Circuits 513 14.2 7400 TTL 518 14.3 TTL Parameters 520 438 476 512 Contents 14.4 14.5 14.6 14.7 14.8 14.9 14.10 14.11 14.12 14.13 TTL Overview 528 Open-collector Gates 530 Three-state TTL Devices 532 External Drive for TTL Loads 534 TTL Driving External Loads 537 74COO CMOS 538 CMOS Characteristics 541 TTL-to-CMOS Interface 544 CMOS-to-TTL Interface 546 Current Tracers 548 Summary 550 Glossary 551 Problems 552 15. A!)piications 15.l Multiplexing Displays 559 15.2 Frequency Counters 565 15.3 Time Measurement 570 15.4 Using the ADC0804 571 15.5 Microprocessor-compatible AID Converters 577 15.6 Digital Voltmeters 585 Summat)' 591 Problems 591 16. A Simple Computer Design 16.1 Building Blocks 594 16.2 Register Transfer Language 597 16.3 Execution oflnstructions, Macro and Micro Operations 599 16.4 Design of Control Unit 602 16.5 Programming Computer 605 Summary 612 Glossary 612 Problems 613 Appendix 1: Binary-Hexadecimal-Decimal Equivalents 615 Appendix 2: 2's Complement Representation 621 Appendix 3: TTL Devices 625 Appendix 4: CMOS Devices 628 Appendix 5: Codes 630 Appendix 6: BCD Codes 633 Appendix 7: Overview of IEEE Std. 91-1984, Explanation of Logic Symbols Appendix 8: Pinout Diagrams 643 Appendix 9: Answers to Selected Odd-Numbered Problems 647 Index 558 593 638 672 Preface to the Seventh Edition (SIE) The seventh edition of Digital Principles and Applications continues with the upgradation of the work started in its previous edition. The job was to build upon the strengths of one of the best introductory and authentic texts in the field of Digital Electronics-its lucid language, down-to-earth approach, detailed analysis and ready-to-use information for laboratory practices. The sixth edition sought improvement primarily by (i) strengthening the design or synthesis aspect that included advanced material, such as a simple computer design, and (ii) incorporating many new topics like Hardware Description Language, Asynchronous Sequential Circuit, Algorithm State Machine chart, Quine-McClusky algorithm, Look Ahead Carry Adder, etc. The tremendous response to the improvements made in the sixth edition from the academic community prompted us to work on their suggestions and come out with this seventh edition. NEW TO THIS EDITION The seventh edition has been revised extensively and restructured to emphasize new and important concepts in Digital Principles and Applications. This edition increases the depth and breadth of the title by incorporating latest infonnation on existing topics like BooleanAlgebra, Schmitt Trigger, 555 Timer, Edge Triggering, Memory Cell, Computer Architecture, and also introduces new topics like Noise Margin, Error Detection and Correction, Universal Shift Register and Content Addressable Memory. The most notable change in this edition is the inclusion of two completely new features-problem solving by multiple methods and laboratory experiments-that will enable the student community develop deeper understanding of the application side of digital principles. Problem solving by multiple methods help students in understanding and appreciating different alternatives to reach a solution, without feeling stuck at any point of time. Laboratmy experiments facilitate experimentation with different analysis and synthesis problems using digital integrated circuits (IC). Each experiment describes its aim, a short reference to theory, apparatus required and different work elements. THE BASIC FEATURES The new edition retains its appeal as a complete self-study guide for a first-level course on Digital Logic and Digital Circuits. It will serve the purpose of a textbook for undergraduate students of CSE, ECE, EEE, Electronics and Instrumentation and IT. It will also be a valuable reference for students of MCA, BCA, DOEACC 'A' Level, as well as BSc/MSc (Computer Science/IT). Preface to the Seventh Edition (SIE) The key features are: >>>>>- Presence of various applications and lab experiments considering the common digital circuit design employed in industries (e.g., LCD display and ADC0804 operation). In-depth coverage of important topics like clock and timing circuits, DIA-AID conversion, register, counters and memory. Tutorial-based approach with section-end self test questions and problem solving through various methods. Useful discussion on TTL and CMOS devices and pin diagrams Rich Pedagogy • 180 Solved Examples • 290 Section-end Problems • 500 Chapter-end Problems COMPREHENSIVE WEBSITE An important addition to this title is the accompanying website- dpa7, designed to be an exhaustive Online Learning Centre (OLC). This website contains the following: For Students • Downloadable codes for HDL examples in the book • Supplementary Reading material Besides Quine-McClusky code and HDL examples, additional information and discussion on various supplementary materials like five-variable Karnaugh Map and Petrick's Algorithm will be available here. Regular updates on different topics of Digital Electronics will be posted to keep the reader informed about recent changes in this field. For Instructors Instructors who have adopted this textbook can access a password-protected section that offers the following resources. • Solution manual • Chapterwise PowerPoint slides with diagrams and notes ACKNOWLEDGEMENTS I would like to acknowledge the inspiration and support I received from Prof. B N Chatterji (Retd.), Prof. RN Pal (Retd.), Prof.RV Rajakumar, Prof. A Chakraborty, Prof.PP Chakraborty, Prof. D Datta, Prof. S Bannerjee, Prof. P K Biswas, Prof. S S Pathak, Prof. S Mukhopadhyaya, Prof. AS Dhar, Prof. I Chakraborty, Prof. P Mandal, Prof. S Mahapatra, Prof. T K Bhattacharya, and all my faculty colleagues at Indian Institute of Technology (IIT) Kharagapur, especially, from the Department of Electronics and Electrical Communication Engineering and GS Sanyal School of Telecommunications. But for the paucity of space, I would have named many others. Preface to the Seventh Edition (SIE) Thanks are due to my research students-Mr S Ari, Mr Md Sahidullah, Mr Israj Ali, and Mr A Manda! for their contribution at different stages of development of the edition. I acknowledge the benefit derived from my interaction with different batches of students while teaching the Digital Electronics subject-three years at Institute of Engineering and Management, Kolkata and over six years at IIT Kharagpur. I am grateful to the esteemed reviewers for their encouraging comments and valuable suggestions for this edition. Sunil Mathur Maharaja Agrasen Institute of Technology, New Delhi VKumar Maharaj a Surajmal Institute of Technology, New Delhi Bijoy Bandopadhyay University College of Science & Technology, Kolkata Anita Kanavalli MS Ramaiah Institute of Technology, Bangalore I also thank the entire team of Tata McGraw Hill Education, more specifically Vibha Mahajan, Shalini Jha, Ashes Saha, Surbhi Suman, Anjali Razdan and Baldev Raj for their support. At this point, I humbly remember all my teachers and my father (late) G N Saha who provided me a great learning environment. I also fondly recollect the contributions in my upbringing ofKharagpur Vivekananda Yuva Mahamandal, Vivekananda Study Circle, IIT Kharagpur Campus and Ramakrishna Mission. I must mention the support I always received from my family-my mother, my parents-in-law, my sisters (specially Chhordi), Chhoto Jamaibabu, and last but not the least, my wife, Sanghita, and daughter, Upasana. The effort behind this work was mine but the time was all theirs. GOUTAM SAHA Feedback Due care has been taken to avoid any mistake in the print edition as well as in the OLC. However, any note on oversight as well as suggestions for further improvement sent at will be gratefully acknowledged (kindly mention the title and author name in the subject line). Also, please report to us any piracy of the book spotted by you. Preface PURPOSE The fifth edition of Digital Principles and Applications is completely recorgnized. It is written for the individual who wishes to learn the principles of digital circuits and then apply them to useful, meaningful design. Thus the title. The material in this book is appropriate for an introductory course in digital logic in either a computer or an electronics program. It is also appropriate for "self-study" and as a "reference" for individuals working in the field. Emphasis is given to the two most popular digital circuit (IC) families-transistor-transistor logic (TTL) and complementary metal oxide silicon (CMOS) logic. Many of these individual I Cs are discussed in detail, and pinouts for more than 60 digital IC chips are summarized in Appendix 8. Standard logic symbols are used along with the new IEEE standard logic. A is given in the appendix. review of the new IEEE symbols I BACKGROUND It is not necessary to have a background in electronics to study this text. A familiarity with Ohm's law and voltage and current in simple de resistive circuits is helpful but not required. If you have no desire to learn about electronics, you can skip Chap. 13. To the extent possible, the remaining chapk:rs are written to be independent of this material. If you have not studied electronics, Chap. 13 will provided the necessary background for you to converse successfully with those who have. Study it any time after Chap. 1. For "old-times" who have studied electronics, Chap. 13 will provide a good review and perhaps a new and valuable point of view. In any case, the material in Chap. 13 will certainly enhance both the knowledge and ability of anyone! ORGANIZATION Each chapter begins with a contents that lists the subjects in each section. The contents listing is followed by a list of chapter objectives. At the end of each chapter section are review questions, called self-tests, which are intended to be a self-check of key ideas and concepts. At the end of each chapter, answers are supplied for the self-tests. A summary and a glossary are provided at the end of each chapter. In any subject area, there are many terms and concepts to be learned. The summary and glossary will provide you with the opportunity to be sure that you understand the exact meaning of these terms, phrases, and abbreviations, The end-of-chapter problems are arranged according to chapter sections. The problems reinforce ideas and concepts presented and allow you to apply them on your own. Solu- Preface tions to selected odd-numbered problems are given at the end of the book. In addition, the appendix contains reference material that will be useful from time to time. LABORATORY EXPERIMENTS A complete set of experiments keyed to this text is available in a laboratory manual, Experiments for Digital Principles. DONALD P. LEACH ALBERT PAUL MALVINO Visual Walkthrough + State machine design using Moore model and Mealy model + Stale transition diagram and preparation of state synthesis table -+ + + + + + + Derivation of design equation from state synthesis table using Karruugh map Circuit implementation: flip-Hop based approach and ROM based approach Use of Algorithm State Machine chart State reduction techniques Anal~is of asynchronous sequential circuit Problems specific to asynchronous sequential circuit Design issues related to asynchronous sequenti;d circuit Design problem normally s1at1s with a word description of input output relation and ends with a d1cuit diagram having sequential and combinatorial logic elements. The word description is first converted to 11 stale transition diagram or Algorithmic State Machine (ASM) chart followed by preparation of stale synthesis tabk. For flip.flop ba.sed implementation, eJtcit:ation tables are used to generate design equations through Kamaugh Map. The final circuit diagram is developed from these design ~tions. 1n Read Only Memory (ROM) based implementation, excitation tables are no! required however; flip.flops are used as dday dements. In this chapter, we show how these techniques can be used in sequt-'tltial circuit design. There are two ditforent approaches. of :s.tate.. machine design called Moore model and Mealy model. In Moore model circuit outpUts, also called primary outputs are generated solely from secondal)' outputs or memory values. In Mealy model circuit inputs, also known as primary inputs combine with memory elements to generate circuit output Both the methods are discussed in detail in this chap1er. In general, sequential logic circuit design refers to sy11chronous clock-triggered circuit because of its design and implementation advantages. Bat there is increasing attention 10 asynchmnous sequential logic Eve,y chapter contains several worked out examples totalling to 180 in the book. Benefits: These will guide the students while understanding the concepts and working out the exercise problems. Eve1y chapter opens with a set of chapter objectives. Benefits: These provide a quick look into the concepts that will be discussed in the chapter. Visual Walkthrough 4,14 • HOIAMl'l:EMENTATION Of DATA PROCESSING ORCUITS We strut with hardware design of multiplexers using Verilog code. The data flow model provides a tlifferenl use of keyword assign in the foml of :ttsigttX'" S? A: B; This statement does following a~signment. If. S'" l,X•• A and if S"" O,X'" B. One can use this st:itement or the logic equation to realize a 2 to l multiplexer shov.n in Fig. 4.2(a) in one of the folkwdng ways. New to this edition, HDL, an interesting development in the field of hardware design, has been introduced. Benefits: The relevant HDL description and codes are weaved into chapters to help students implement and design digital circuits. Programming a PAL A PAL is diffen-nt from a PROM because it has a progr.mumtble AND array and a fixed OR array. For ins.ancc. Fig. ·1-43 shows a PAL with-4 inputs and 4 outputs. The x's on the input side are fusible links, while 1he solid black bullets on the output sid-c are fixed connections. With a PROM programmer, we can bum in the desired fumfamental products, which .ire then OR.ed by the fixed output conn,..--ctions. C D ,7 '"7 ,7 v Fixc.dORamy = = = I Figures are used exhaustively in the text. Benefits: These illustrate the concepts and methods described for better understanding. -= = = I I I I I I I r I I I ,-,n = = = -= = = -= 99 Structure of PAL Visual Wa/kthrough A section called Se(f-Test appears a.fier evet)' section in eve1y chapte1: Benefits: This will help students check their understanding of the concepts discussed in a section before moving on to the next section. Answers to Seif-Tests are given at the end of that chapte1: 2 .and 3; (b) :5 .md5; (c)9and 9, 22, What11t~ the dig1ttl 01:tlflli! ievcb of the cnc,mier in Fig_ t26a if only infll,ll line 6 i11 l1igh? A Brief summa,y is provided at the end of the chapters. Benefits: Summary gives the essence of each chapter in bri-Y Y=notA 1.e. Y=A' sothat,ifA=O,Y=O'=l W andif A=l,Y=l'=O The truth tables in Fig. 2.1 illustrate signal levels that do not change with time. However, almost all digital signals do in fact change with time, as illustrated by the waveforms in Chapter 1 (Sec. 1.2). Here are two examples that illustrate how to use the truth table information with signals that vary with time. A I-kHz square wave drives pin 1 of a 7404 (see Fig. 2.2). What does the voltage waveform at pin 2 look like? Digital Principles and Applications Solution Figure 2.3a shows what you will see on a dual-trace oscilloscope. Assuming you have set the sweep timing to get the upper waveform (pin I), then you would see an inverted square wave on pin 2. If a 500-Hz square wave drives pin 3 of a 7404, what is the waveform on pin 4? Solution Pins 3 and 4 are the input and output pins of an inverter (see Fig. 2.2). A giance at Fig. 2.3b shows the typical waveforms on the input (pin 3) and output (pin 4) of a 7404. Again, the output waveform is the complement of the input waveform, Because of two-state operation, rectangular wavefonns like this are the normal shape of digital signals. Incidentally, a timing diagram is a picture of the input and output waveforms of a digital circuit. Examples of timing diagrams are shown in Figs. 2.3a and b. Pin 1 Pin 3 A B C D E F G H I A C E G I A C E G I Pin4 Pin2 A B C D E F G H I (b) (a) Cfl F.i;:rl::) OR Gates An OR gate has two or more input signals but only one output signal. It is called an OR gate because the output voltage is high if any or all of the input voltages are high. For instance, the output of a 2-input OR gate is high if either or both inputs are high. Figure 2.4a shows the logic symbol of a 2-input OR gate and Fig. 2.4b its truth table. B y 0 I 1 0 1 I 0 A B~y 0 0 I I I (b) (a) (a) OR gate, (b) Truth table In Boolean equation form so that Y = A ORB. 1.e. Y = A + B Y=O+O=O, Y=O+ I= 1, Y= I +O= I and Y= I+ I = 1. The '+' sign here represents logic operation OR and not addition operation of basic arithmetic. Note that in arithmetic 1 + 1 = 2 in decimal and l + 1 = l O in biA B C Y nary number system (Table 1.1 of Chapter I). Binary 0 0 0 0 addition is discussed in detail in Chapter 6. 0 0 I l Three Inputs Figure 2.5 shows a 3-input OR gate. The inputs are A, B, and C. When all inputs are low, Y is low. If A or B or C is high, Y will be high. The truth table summarizes all input possibilities. In equation form, the three input OR gate is represented as: Y = A+B+C. The truth table (Fig. 2.5b) allows us to check that all input possibilities are included. Why? Because every possibility is included when the input entries DA~ C~y 0 0 1 l 0 I I l 0 0 0 1 I 0 I (a) 1 (b) (a) Three-input OR gate, (b) Truth table Digital Logic follow a binary sequence. For example, the first ABC entry is 000, the next is 001, then 010, and so on, up to the final entry of 111. Since all binary numbers are present, all input possibilities are included. Incidentally, the number of rows in a truth table equals 2", where n is the number of inputs. For a 2-input OR gate, the truth table has 22, or 4 rows. A 3-input OR gate has a truth table with 23, or 8 rows, while a 4-input OR gate results in 24, or 16 rows, and so on. An OR gate can have as many inputs as desired. No matter how many inputs, the action of any OR gate is summarized like this: One or more high inputs produce a high output. logic Symbols Figure 2.6a shows the symbol for a 2-input OR gate of any design. Whenever you see this symbol, remember the output is high if either input is high. Shown in Fig. 2.6b is the logic symbol for a 3-input OR gate. Figure 2.6c is the symbol for a 4-input OR gate. For these gates, the output is high when any input is high. The only way to get a low output is by having all inputs low. (c) (b) (a) (d) OR gate symbols: (a) Two-input, (b) Three-input, (c) Four input, (d) Twelve-input When there are many input signals, it's common drafting practice to extend the input side as needed to allow sufficient space between the input lines. For instance, Fig. 2.6d is the symbol for a 12-input OR gate. The same idea applies to any type of gate; extend the input side when necessary to accommodate a large number of input signals. TTL OR Gates Figure 2.7 shows the pinout diagram of a 7432, a TTL quad 2-input OR gate. This digital IC contains four 2-input OR gates inside a 14-pin DIP. After connecting a supply voltage of +5 V to pin 14 and a ground to pin 7, you can connect one or more of the OR gates to other TTL devices. Timing Diagram Figure 2.8 shows an example of a timing diagram for a 2-input OR gate. The input voltages drive pins 1 and 2 of a 7432. Notice that the output (pin 3) is low only when both inputs are low. The output is high the rest of the time because one or more input pins are high. Pin 1 B 7432 D F H Pin 2 Pin3 GND Pinout diagram of a 7432 A B C D E F G H I A L D E H I L Timing diagram Digital Principles and Applications Work out the truth table for Fig. 2.9a. Solution With two input signals (A and B), four input cases are possible: low-low, low-high, high-low, and highhigh. For convenience, let L stand for low and Hfor high. Then, the input possibilities are LL, LH, HL, and HH, as listed in Fig. 2.9b. Here is what happens for each input possibility. CASE l A is !ow and Bis low. With. both input voltages in the low state, each inverter has a high output This means that the OR gate has a high output, the first entry of Fig. 2.9b. CASE 2 A is low and B is high. With these inputs the upper inverter has a high output, while the lower inverter has a low output. Since the OR gate still has a high input, the output Y is high. CASE 3 A is high and B .is low. Now, the upper.inverter has a low output and the lower inverter has a high output Again, the OR gate produces a high output, so that Y is high. CASE 4 A is high and B is high. With both inputs high, each inverter has a low output. This time, the ORgate has all inputs in the low state, so that Y is low, as shown by the final entry of Fig. 2.9b. 7404 A 2 y B (a) A B y L L H H L H H H H L L H {b) Logic circuit and truth table of Example 2.3 Incidentally, the circuit of Fig. 2.9a uses only one-third of a 7404 and one-fourth ofa 7432. The other gates in these digital ICs are not connected, which is all right because you don't have to use all of the available gates. AND Gates The AND gate has a high output only when all inputs are high. Figure 2.1 Oa shows a 2-input AND gate. The truth table (Fig. 2.1 Ob) summarizes all input-output possibilities for a 2-input AND gate. Examine this table carefully and remember the following: the AND gate has a high output only when A and B are high. In other words, the AND gate is an all-or-nothing gate; a high output occurs only when all inputs are high. This truth table uses Is and Os, where 1 = Hand O = L. In Boolean equation fonn so that, A B~y (a) B Y 0 0 0 0 1 0 1 0 0 1 I l (b) (a) Two-input AND gate, (b) Truth table Y =A AND B, i.e. Y=A.B or Y=AB Y = 0.0 = 0, Y = 0.1 = 0, Y = 1.0 = 0 and Y = 1.1 = 1 The '.' sign here represents logic AND operation and not multiplication operation of basic arithmetic though the result are same for both. Digital Logic Figure 2.11 a shows a 3-input AND gate. The inputs are A, B, and C. When all inputs are low, Y is low. If even one input is low, Y is in the low state. The only way to get a high output is to raise all inputs to the high state (+5 V) The truth table (Fig. 2.11 b) summarizes all input-output possibilities. In equation form, the three input AND gate is represented as: Y=A.B.C=ABC. Three Inputs C~y A B C y 0 0 0 0 0 0 I I 0 I 0 I 0 0 0 0 0 0 0 I 1 0 I I 0 0 0 I Logic Symbols Figure 2.12a shows the symbol (a) (b) for a 2-input AND gate of any design. Shown in Fig. 2.12b is the logic symbol for a 3-input (a) Three-input AND gate, AND gate. Figure 2.12c is the symbol for a 4(b) Truth table input AND gate. Remember: For any of these gates, the output is high only if all inputs are high. As before, it's common drafting practice to extend the input sides when there are many input signals. For instance, Fig. 2.12d is the symbol for a 12-input AND gate. (a) (b) (d) (c) AND gate symbols: (a) Two-input, (b) Three-input, (c) Four-input, (d) Twelve-input Figure 2.13 shows the pinout diagram of a 7408, a TTL quad 2-input AND gate. This digital IC contains four 2-input AND gates. After connecting a supply voltage of +5V to pin 14 and a ground to pin 7, you can connect one or more of the AND gates to other TTL devices. TTL AND gates are also available in triple 3-input and dual 4-input packages. (See Appendix 3 for pinout diagrams.) TTL AND Gates Figure 2.14 shows an example of a timing diagram for a 2-input AND gate. The input voltages drive pins 1 and 2 of a 7408. Notice that the output (pin 3) is high only when both inputs are high (between C and D, G and H, etc.). The output is low the rest of the time. Timing Diagram Pin l F B 7408 Pin2 A Pin3 GND Pinout diagram of a 7408 B C D F G H I H C D G H Timing diagram Digital Principles and Applications Work out the truth table for Fig. 2.15a. A y A 0 0 B 0 A' B' Y=A~B' r I 0 1 1 1 0 0 1 I 0 1 0 1 0 0 0 (b) (a) logic circuit and truth table of Example 2.4 Solution We get the final truth table here in slightly different way. Consider, one logic gate at as shown in Fig. 2:15b; The NOT gate connected to A gives A I at its output and is shown in column 3. The NOT gate connected to B gives B' at its output and is shown in column 3. Finally, the 4'" column shows OR operation on column 3and 4 to give the final output Y. Here is what happens for each input possibility. CASE 1 A is low and B is low. With both input voltages in the low state, each inverter has a high output This means the AND gate has a high output, the first entry of Fig. 2.16. CASE 2 A is low and B is high. With these inputs the upper inverter has a high output, while the lower inverter has a low output Since the AND gate produces a low output, Y is low. CASE 3 A is high and B is low. Now, the upper inverter has a low output and the lower output Again, the AND gate produces a low output, so Y is low. inverter has a A L L y B L H H L H L L H H1L CASE 4 A is high and B is high. With both inputs high, each inve1ter has a low output Again, the AND gate has a low as shown bythe final.entry of Fig. 2.16. Note that input-output relations described i11 Fig. 2.15b and Fig. 2.16 are same. What is the Boolean equation for the logic circuit of Fig. 2. l 7a? Solution This circuit is called an AND-OR network because input AND gates drive an output OR gate~ The intermediate outputs are =AB =CD The :final output is f8 Y3 +Y,, Y=AB+CD An equation in this form is referred to as a sum-ofproducts equation. AND-OR networks always produce sum-ofproducts equations. Write the Boolean equation for Fig. 2.17b. Solution Thislogic circuit u,ic.,u«;u an OR-AND network because input OR gates drive an output AND gate. The and Y11 = C + D. intem1ediate outputs are Y8 =A Digital Logic The :final output is or B)(C+D) A B y y C D (a) AND-OR, {b) OR-AND network As shown in this equation, parentheses may be used to indicate a logical product (ANDing), Also notice that the final answer is a product of sums. OR-AND networks always produce product-of-sums equations. What is the logic circuit whose Boolean equation is Y= ABC+ABC Solution This is ,i ~um-of-products equati9n with some of the inputs in complemented fonn. Figure.2.18a shows an .AND-OR circuit with the foregoing Boolean equation. upper AND gate. produces a logical product of =ABC The lower AND gate proi:luces A 7411 7411 A -----~•.:;:,o.----2,,.t__ B ----,-+---------+ B 13 C y 3 A 3 B 4 4 5 C (a) (b) (a) Intermediate, (b) Final logic circuit of Example 2.7 I. A system in which H= 1 and L = 0 is (positive, negative) logic. 2. A gate whose output is H if any input is His an gate. 3. A gate whose output is H only when all inputs are His an ____ gate. 7432 Digital Principles and Applications The final output therefore equals the sum of the f 12 and Y6 products: Y= ABC+ABC The complemented inputs A and B may be produced by other circuits (discussed later). Alternatively, inverters on the A and B input lines may produce the complemented variables, as shown in Fig. 2.I8b. 4. Write an expression for an inverter, or NOT gate equivalent to Y = not A. 5. Write a Boolean expression for an OR gate having A and Bas inputs and Yas the output. 6. Write the Boolean expression for an AND gate with A and B as inputs and Y as the output. This example illustrates one method oflogic design. Whenever you are given a sum-of-products equation, you can draw the corresponding AND-OR network using AND gates to produce the logical products and an OR gate to produce the sum. 2.2 UNIVERSAL LOGIC GATES-NOR, NANO In the previous section we have seen how AND, OR and NOT gates can be connected together to realize any logic function. Here, we address an interesting question. Is it possible to use only one type of gate for this purpose? If possible, one needs to procure only one type of gate for his design. And more importantly, fabrication of Integrated Circuit that perfonns a logic operation becomes easier when gate of only one kind is used. Gates, which can perform this task, are called universal logic gates. Clearly, basic gates like AND, OR and NOT don't fit into this category for the simple reason that conversion among themselves itself are not possible. As for example, one cannot gate OR operation by using any number or combination of AND gates. In this section, we discuss two universal logic gates NOR and NAND. NOR Gates The logic circuit of Fig. 2.19a used to be called a NOT-OR gate because the output is Y=A+B Read this as "Y equals NOT A ORB" or "Y equals the complement of A ORB." Because the circuit is an OR gate followed by an inverter, the only way to get a high output is to have both inputs low, as shown in the truth table of Table 2.1. NOR Gate Symbol The logic circuit of Fig. 2.19a has become so popular that the abbreviated symbol of Fig. 2.19b is used for it. The bubble (small circle) on the output is a reminder of the inversion that takes place after the ORing. Furthermore, the words NOT-OR are contracted to the word NOR. So from now, we will call the circuit a NOR gate and will use the symbol of Fig. 2.19b. Whenever you see this symbol, remember that the output is NOT the OR of the inputs. With a NOR gate, all inputs must be low to get a high output. If any input is high, the output is low. Digital Logic ~=D-{)»-r ~=[>-r ~=§-r (b) (a) (c) 2 3 5 7402 4 6 8 10 9 11 7 13 12 (GND) (e) (d) NOR logic gate Figure 2.19c shows the new IEEE rectangular symbol for the NOR gate. The small triangle on the output is equivalent to the bubble used on the standard symbol. The indicator~ inside the box means "if one or more of the inputs are high, the output is high." The 7402 is a quad 2-input NOR gate in a 14-pin DIP as illustrated in Fig. 2.19d. The new rectangular symbol for the 7402 is shown in Fig. 2.19e. NOR Gate A B y 0 0 I l 0 I 0 I 0 0 0 Bubbled AND Gate Figure 2.20a shows inverters on the input lines of an AND gate. This logic circuit is often drawn in the abbreviated form shown in Fig. 2.20b. (a) The bubbles on the inputs are a reminder of the inversion that takes place before the AND operation. We will refer to the abbreviated drawing of (b) Fig. 2.20b as a bubbled AND gate. We have already analyzed this circuit in Example 2.4 and obtained (a) AND gate with inverted its truth table as shown in Fig. 2.15b. We find that inputs, (b) Equivalent symbol output Y and inputs A, B are identical for bubbled AND gate and NOR gate. Therefore, these two circuits are equivalent and thus interchangeable. Given any logic circuit with NOR gates, we can replace it by bubbled AND gates and converse is also true. De Morgan's First Theorem The Boolean equation for Fig. 2.19b is Y=A+B The Boolean equation for Fig. 2.20b is Y=AB Digital Principles and Applications The first equation describes a NOR gate, and the second equation a bubbled AND gate. Since the outputs are equal for the same inputs, we can equate the right~hand members to get (2.1) A+B = AB This identity is known as De M01gan ~, first theorem. In words, it says the complement of a sum equals the product of the complements. This can also be proved by comparing the truth tables shown in Fig. 2.4(b) and NOR gate truth table of Table 2.1. A similar exercise that compares truth tables of three input NOR gate and three input bubbled AND gate show they are identical and we can write, (A+ B + C)' = A'B'C'. Note that this equivalence can be extended to gates or circuits for larger number of inputs, too. Universality of NOR Gate Figure 2.21 shows how all other logic gates can be obtained from NOR gates. To get a NOT gate we tie inputs of NOR gate together (Fig. 2.21 a) so that there is only one input to the circuit. If input is 0, then both the inputs to NOR gate are 0. Following NOR gate truth table (Table 2.1) we see output now is l. Similarly, if input is 1, both the inputs to NOR gate are 1 that gives output 0. Therefore output of circuit, shown in Fig. 2.2 la is complement of its input and thus gives NOT operation. A A~A;~A.+B B (a) (b) (c) Universality of NOR gate (a) NOT from NOR, (b) OR from NOR, (c) AND from NOR Figure 2.21 b shows how to get OR circuit using only NOR gates. The first NOR gate performs usual NOR operation while second NOR gate performs as NOT gate and inverts the NOR logic to OR. To understand how we get AND circuit using only NOR gates (Fig. 2.21c) let us refer to example 2.3. The configuration is similar except the output there is generated from OR and here from NOR and of course the NOT gates are replaced by NOR equivalent. Since NOR gate is NOT operation followed by OR we invert the output of example 2.3, shown in Fig. 2.9b to get output of this circuit. Thus output of circuit in Fig. 2.2 lc is high only when both the inputs are high and it functions like an AND gate. The above equivalences can be proved simply, by applying Boolean theorems and we'll discuss those theorems in next chapter. Since, we can perform all the Boolean operations using only NOR gates it is termed as universal logic gate. Eye of the Beholder Which brings us to a principle. Truth tables, logic circuits, and Boolean equations are different ways of looking at the same thing. Whatever we learn from one viewpoint applies to the other two. If we prove that truth tables are identical, this immediately tells us the co1Tesponding logic circuits are interchangeable, and their Boolean equations are equivalent. When analyzing, we generally start with a logic circuit, construct Digital Logic its truth table, and summarize with the Boolean equation. When designing, we often startwith a truth table, generate a Boolean equation, and arrive at a logic circuit. A 7402 is a quad 2-input NOR gate. This TTL IC has four 2-input NOR gates in a 14-pin DIP as shown in Appendix 3. What is the Boolean equation for the output of Fig. 2.22a? Solution The AND gates produce AB and CD. These are ORed to get AB+ CD. The final inversion gives Y=AB+CD The circuit of Fig. 2.22a is known as anAND-OR-INVERT network because it starts withANDing, follows with ORing, and ends with INVERTing. The AND-OR-INVERT network is available as a separate TTL gate. For instance, the 7451 is a dual 2-input 2-wide AND-OR-INVERT gate, meaning iwo networks like Fig. 2.22a in a single 14-pin TTL package. Appendix 3 shows the pinout diagram. Figure 2.22b shows how we can use half of a7451 to produce the same output as the circuit of Fig. 2.22a. 7408 7451 A B A B y y C D C D (b) (a) AND-OR-INVERT network Prove that Fig. 2.23c is logically equivalent to Fig. 2.23a. Solution De Morgan's first theorem says we can replace the final NOR gate ofFig. 2.23a by a bubbled AND gate to get the equivalent circuit of Fig. 2.23b. If you invert a signal twice, you get the original signal back again. Put

another way, double inversion has no effect on the logic state; double invert a low and you still have a low; double-invert a high and.you still have a high. Therefore, each double inversion in Fig. 2.23b (a pair of bubbles on the same signal line) cancels out, leaving the simplified circuit of Fig. 2.23c. Therefore, Fig. 2:23a and Fig. 2.23c are equivalent or interchangeable. Why would anyone want to replace Fig, 2.23a by 2.23c? Suppose your shelves are full of AND gates and OR gates .. Ifyou have just run out of NOR gates and you are trying to build a NOR-NOR network like Fig. 2.23a, you can connect the OR-AND circuit of Fig. 2.23c because it produces the same output as the original circuit. In general, this idea applies to any circuit that you can rean-ange with De Morgan's theorem. You can build whichever equivalent circuit is convenient. A B A B y y y C D (a) (b) (c) Equivalence among logic circuits: Example 2.9 Digital Principles and Applications What is the truth table for the NOR-NOR circuit of Fig. 2.23a? Convert Table 2.2 into a timing diagram. Solution ... In tabl~ 2.2, IDflltD change;, statesfon~ach entry, input G'changes states every otl.ler entry, inpufB every fourth entry, andjn.putA every eighth. eμtry. figure 2.24 shows how to draw the table in thef~llll ~f a tinli:11g diagram. First, notice jhalthe transitions on i~put D are l, 2, 3, and. so on. Notic~. that input D. changes. state5 each tr.lt:tsition, inp~tC eye137.?thertransition, input B . every fourth transition, and.illpμtA~very . eighthtransition.. ·Toagree wit~ Jhe tl1.lth tllbJf,.Qutput Yis low up to transition5, high between 5 and. 8; low between 8 and 9, and so forth. trutb D 0 l 0 1 0 1 0 1 0 1 0 1 0 1 0 Digital Logic NANO Gates Originally, the logic circuit of Fig. 2.25a was called NOT-AND gate because the output is Y=AB Read this as "Y equals NOT A AND B" or "Yequals the complement of A AND B." Because the circuit is an AND gate followed by an inverter, the only way to get a low output is for both inputs to be high, as shown in the truth table of Table 2.3. ;D-{>-y (a) ;u-y ~=B-y (b) (c) 1 3 2 6 5 7 9 IO 8 12 13 14 (e) NANO logic gate NANO-Gate Symbol The logic circuit of Fig. 2.25a has become so popular that the NANO Gate abbreviated symbol of Fig. 2.25b is used for it. The bubble on the output reminds us of the inversion after the ANDing. Also, y the words NOT-AND are contracted to NAND. Whenever you 0 1 see this symbol, remember that the output is NOT the AND of 1 l the inputs. With a NAND gate, all inputs must be high to get a 0 1 low output. If any input is low, the output is high. 0 Figure 2.25c shows the new IEEE rectangular symbol for the NAND gate. The small triangle on the output is equivalent to the bubble used on the standard symbol. The indicator"&" inside the box means "the output is high only when all inputs are high." The 7400 is a quad 2-input NAND gate in a 14-pin DIP as illustrated in Fig. 2.25d. The new rectangular symbol for the 7402 is shown in Fig. 2.25e. Bubbled OR Gate Figure 2.26a shows inverters on the input lines of an OR gate. The circuit is often drawn in the abbreviated form shown in Fig. 2.26b, where the bubbles represent inversion. We will refer to the abbreviated drawing of Fig. 2.2b as a bubbled OR gate. We have already analyzed this circuit in Example 2.3 and obtained its truth Digital Principles and Applications table in Fig. 2.9b. We see that output Y and inputs A, Bare identical for bubbled OR gate and NAND gate. Therefore, these two circuits are equivalent and thus interchangeable. Given any logic circuit with NOR gates, we can replace it by bubbled AND gates and converse is also true. A y B (a) ~=D-r De Morgan's Second Theorem (b) The Boolean equation for Fig. 2.24b is (a) OR gate with inverted inputs, (b) Equivalent symbol Y=AB The Boolean equation for Fig. 2.25b is Y=X +B The first equation describes a NAND gate, and the second equation a bubbled OR gate. Since the outputs are equal for the same inputs, we can equate the right-hand members to get (2.2) AB= A +B This identity is known as De Morgans second theorem. It says the complement of a product equals the sum of the complements. This can also be proved by comparing the truth tables shown in Fig. 2.3(b) and NAND gate truth table of Table 2.2. A similar exercise that compares truth tables of three input NAND gate and three input bubbled OR gate show they are identical and we can write, (A.B.C)' =A'+ B' + C'. Note that this equivalence can be extended to gates or circuits with any number of inputs. Universality of NANO Gate Figure 2.27 shows how all other logic gates can be obtained from NAND gates and why it is called a universal logic gate. Figure 2.27a shows how we tie inputs ofNAND gate together (as we had done in case of NOR gate) to get a NOT gate that has only one input. If input is 0, then both the inputs to NAND gate are. 0. Following NAND gate trnth table (Table 2.3) we see output now isl. Similarly, if input is 1, both the inputs to NAND gate are 1 that gives output 0. Therefore output of circuit, shown in Fig. 2.27a is complement of its input and thus gives NOT operation. Figure 2.27b shows how we get AND circuit using only NAND gates. The second NAND gate performs as a NOT gate and inverts the NAND logic offirst NAND gate to AND logic. A A·~A;~A.B B (a) (b) (c) Universality of NAND gate: (a) NOT from NAND, (b) AND from NAND, OR from NAND (c) Digital Logic ==c=>- =D- =D- =r>- (a) (b) ¢=} =D- ¢=} ¢=} =CJ- =r=>- =[)- ¢=} {c) (d) Useful logic equivalences To obtain OR logic using NAND gate we compare Fig. 2.27c circuit with Fig. 2.15a. The later gives NOR logic and has AND gate at output. The present circuit has NAND gate at output and thus inverts the output of previous circuit, from NOR to OR. TTL NANO Gates The NAND gate is the backbone of the 7400 TTL series because most devices in this family are derived from it. Because of its central role in TTL technology, the NAND gate has become the least expensive and most widely used TTL gate. Furthermore, the NAND gate is available in more configurations than other gates, as shown in Table 2.4. Notice that the NAND gate is available as a 2-, 3-, 4-, or 8-input gate. The other gates have fewer configurations, with the OR gate available only in 2-input fonn. Standard TTL Gates Type Quad 2-lnput Triple 3-lnput Dual 4-lnput Single 8-lnput NAND NOR AND OR 7400 7402 7408 7432 7410 7427 74ll 7420 7425 7421 7430 Prove that Fig. 2.29c is logically equivalent to Fig. 2.29a. A B y C D (a) Equivalence of logic gates: Example 2.12 Solution Ile.Mor~an's second theorem says we can replace the final NAND gate of Fig. 2.29a by a bubbled OR gate to getthe equivalent circuit ofFig. 2.29b. Each double inversion in Fig. 2.29b cancels out, leaving the simplified circuit of Fig. 2.29c .. Therefore, Figs. 2.29a and 2.29c are equivalent Incidentally, most people find Fig. 2.29b easy to analyze because they learn to ignore the double inversions and see only the simplified AND-OR circuit of Fig. 2.29c. For this reason, if you build a NAND-NAND Network like Fig. 2.29a, you can draw it like Fig. 2.29b. Anyone who sees Fig. 2.29b on a schematic diagram will know it is two Digital Principles and Applications input NAND gates driving an output NAND gate. Furthermore, when troubleshooting the circuit, they can ignore the bubbles and visualize the easy-to-analyze AND-OR circuit of Fig. 2.29c. What is the truth table for the NAND-NAND circuit of Fig. 2.29a? Solution Let us analyze the equivalent circuit of Fig. 2.29c because it is simpler to work with. Table 2.5 lists every possibility starting with all inputs low and progressing to all inputs high. By analyzing each input possibility, we can determine the resulting output Fo.r fostance, when all inputs llt'e 1owin Fig. 2,29c, ho.th AND gates have low outputs, so the OR gate produces a low output. This is the first entry of Table 2.5. Proceeding like this, we can arrive at the output for the remaining possibilities ofTable2.5, Show a timing diagram for the NAND-NAND circuit of Fig. 2.29a. Solution All you have to do is convert the low-high states of Table 2.5 into low-high waveforms like First, notice that. the. traJJ.sitions .on. input a.re. numbered l, 2, 3, and S() on. Input D chan~es states each .transition, transition, and input A every eighth transiti~n. To agree with the input C every other transition, input 8 ~very truth table, output Yis low up to transition 3,.highbetween 3 and 4, lowl)etween 4 and 7, and so forth. D fourth NANO-NANO Circuit A 8 C D r. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 l 0 I 1 1 0 0 0 0 0 0 0 1 0 0 0 l 0 0 0 1 l I 1 1 l I I 1 I 1 0 0 l 0 I 0 1 0 l 1 0 0 l 0 1 0 0 1 l I I l 0 1 0 0 A l I 1 1 l 0. Write the Boolean expression for a 2 input NANJJ gate. 11. 'Write De}forgan's second theorem. . < .•· 12. Wha~ symbol is used inside the IEijErectangular box to define a. NANI) gate? Timing diagram Digital Logic 2.3 AND-OR-INVERT GATES Figure 2.31a shows an AND-OR circuit. Figure 2.31b shows the De Morgan equivalent circuit, a NANDNAND network. In either case, the Boolean equation is Y=AB+CD A B y C A B y (a) y C C D D A B D (b) (c) (a) AND-OR circuit, (b) NANO-NANO circuit, (c) AND-OR-INVERT circuit Since NAND gates are the preferred TTL gates, we would build the circuit of Fig. 2.31 b. As you know, NAND-NAND circuits like this are important because with them you can build any desired logic circuit. TTL Devices AND-OR circuits are not easily derived from the basic NAND-gate design. But it is easy to get an ANDOR-INVERT circuit as in Fig. 2.3 lc. A variety of circuits like this are available as TTL chips. Because of the inversion, the output has the equation shown below. (2.3) Y= AB+CD Table 2.6 lists the AND-OR-INVERT gates available AND-OR-INVERT Gates in the 7400 series. In this table, 2-wide means two AND gates across, 4-wide means four AND gates across, and Description Device so on. For instance, the 7454 is a 2-input 4-wide ANDDual 2-input 2-wide 7451 OR-INVERT gate as in Fig. 2.32a; each AND gate has 2-input 4-wide 7454 two inputs (2-input), and there are four AND gates (4Dual 2-3-input 2-wide 7459 wide). Figure 2.32b shows the 7464; it is a 2-2-3-4-input 2-2-3-4-input 4-wide 7464 4-wide AND-OR-INVERT gate. Connecting the output of a 2-input 2-wide AND-OR-INVERT gate to an inverter will give us the same output as an AND-OR circuit. (a) (b) Examples of AND-OR circuits Digital Principles and Applications Expandable AND-OR-INVERT Gates The widest AND-OR-INVERT gate available in the 7400 series is 4-wide. What do we do when we need a 6- or 8-wide circuit? One solution is to use an expandable AND-OR-INVERT gate. Figure 2.33 shows the logic symbol for an expandable AND-OR-INVERT gate. There are two additional inputs, labeled bubble and arrow. Table 2.7 lists the expandable AND-OR-INVERT gates in the 7400 series. A B y Expandable AND-OR-INVERT Gates C D Bubble----~ Arrow - - - - - ~ Expandable AND-OR-INVERT gate Device Description 7450 7453 7455 Dual 2-input 2-wide 2-input 4-wide 4-input 2-wide Expanders What do we connect to the arrow and bubble inputs of an expandable gate? We connect the output of an expander as in Fig. 2.34a. Connect bubble to bubble and arrow to arrow. Visualize the outputs of Fig. 2.34a connected to the arrow and bubble inputs of Fig. 2.33. Figure 2.34b shows the logic circuit. This means that the expander outputs are being ORed with the signals of the ANDOR-INVERT gate. In other words, Fig. 2.34b is equivalent to the AND-OR-INVERT circuit ofFig. 2.34c. y ~Bubble =L.__;- Arrow (a) (b) y y (c) (d) (a) Expander, (b) Expander driving expandable AND-OR-INVERT gate, (c) ANDOR-INVERT circuit, (d) Expandable AND-OR-INVERT with two expanders Digital Logic We can connect more expanders. Figure 2.34d shows two expanders driving the expandable gate. Now we have a 2-2-4-4-input 4-wide AND-OR-INVERT circuit. The 7460 is a dual 4-input expander. The 7450, a dual expandable AND-OR-INVERT gate, is designed for use with up to four 7460 expanders. This means that we can add two more expanders in Fig. 2.34d to get a 2-2-4-4-4-4-input 6-wide AND-OR-INVERT circuit. 13. When we speak ofan AND-OR-INVERT gate, what is the meaning of2--wide? 14. What is the purpose of using an expander with an AND-OR..JNVERT gate? 2.4 POSITIVE AND NEGATIVE LOGIC Up to now, we have used a binary O for low voltage and a binary 1 for high voltage. This is called positive logic. People are comfortable with positive logic because it feels right. But there is another code known as negative logic where binary O stands for high voltage and binary 1 for low voltage. Even though it seems unnatural, negative logic has many uses. The following discussion introduces some of the terminology and concepts for both types of logic. Positive and Negative Gates An OR gate in a positive logic system becomes an AND gate in a negative logic system. Why? Look at the gate of Fig. 2.35. We have been calling it an OR gate. This is correct, provided we are using positive logic. Table 2.8 shows the tn1th for the gate of Fig. 2.35, no matter what you call it. That is, if either input is high in Fig. 2.35, the output is high. Positive OR A Negative AND Meaning ofsymbol depends on whether you use positive or negative logic Low High High Low High Low High In a positive logic system, binary O stands for low and binary 1 for high. So, we can convert Table 2.8 to Table 2.9. Note that Y is a 1 if either A or B is 1. This sounds like an OR gate. And it is, because we are using positive logic. To avoid ambiguity, we can call Fig. 2.35 a positive OR gate because it performs the OR function with positive logic. (Some data sheets describe gates as positive OR gate, positive AND gate, etc.) In a negative logic system, binary 1 stands for low and binary O for high. With this code, we can convert Table 2.8 to Table 2.10. Now, watch what happens. The output Y is a 1 only when both A and B are 1. This sounds like an AND gate! And it is, because we are now using negative logic. In other words, gates are defined by the way they process the binary Os and 1s. If you use binary 1 for low voltage and binary O for high voltage, then you liave to refer to Fig. 2.35 as a negative AND gate. As you see, the gate of Fig. 2.35 always produces a high output if either input is high. But what you call it depends on whether you see positive or negative logic. Use whichever name applies. With positive logic, call it a positive OR gate. With negative logic, call it a negative AND gate. Digital Principles and Applications y 0 0 0 In a similar way, we can show the truth table of other gates with positive or negative logic. By analyzing the inputs and outputs in terms of Os and ls, you find these equivalences between the positive and negative logic: Positive OR Positive AND Positive NOR Positive NAND negative AND H negative OR H negative NAND H negative NOR H Table 2.11 summarizes these gates and their definitions ill terms of voltage levels. These definitions are always valid. If you get confused from time to time, refer to Table 2.11 to get back to the ultimate meaning of the basic gates. Voltage Definitions of Basic Gates ·•· ··.·· Gate Definition ·. . . • J:>ositivl'l O!Vn,egatjveAND .E'ositive AND/negative OR Positive NOR/negative NAND P?sitiye NAND/negative-NOR .•. Output is high if any input is high . Output is high when all inputs are high. Output is low if any input is high. Output is low when all inputs are high. Assertion-level logic Why do we even bother with negative logic? The reason is related to the concept of active-low signals. For instance, the 74150 multiplexer has an active-low input strobe; this input turns on the chip only when it is low (negative true). This is an active-low signal; it causes something to happen when it is low, rather than high. As another example, the 74154 decoder has 16 output lines; the decoded output signal is low (negative true). In other words, all output lines have a high voltage, except the decoded output line. Besides TTL devices, microprocessor chips like the 8085 have a lot of active-low input and output signals. Many designers draw their logic circuits with bubbles on all pins with active-low signals and omit bubbles on all pins with active-high signals. This use of bubbles with active-low signals is called assertion-level logic. It means that you draw chips with the kind of input that causes something to happen, or with the kind of output that indicates something has happened. If a low input signal turns on a chip, you show a bubble on that input. If a low output is a sign of chip action, you draw a bubble on that output. Once you get used to assertion-level logic, you may prefer drawing logic circuits this way. One final point. Sometimes you hear expressions such as "The inputs are asserted" or "What happens when the inputs are asserted?" An input is asserted when it is active. This means it may be low or high, depending on whether it is an active-low or active-high input. For instance, given a positive AND gate, all inputs must be asserted (high) to get a high output. As another example, the STROBE input ofa TTL multiplexer must be Digital Logic asserted (low) to turn on the multiplexer. In short, you can equate the word assert with activate. You assert, or activate, the inputs of a gate or device to get something to happen. Here are some ideas that you should try to remember: 1. Positive true always represents a high voltage, and negative true always represents a low voltage. 2. If possible, draw basic gates with bubbles on active-low signal lines. 3. When a signal is active-low, use an overbar as a reminder that the signal voltage is negative true when the underlying statement is true. (a) The number stored in a register may be zero (all bits low). Show how to detect this condition. (b) What change in (a) will detect presence of the word 10110101 in the 8-bit register? Solution a Satlle as {a) Figure 236 shows design using assertion-level logic. The bits go to a bubbled AND gate (the positive NOR gate). When all the bits are low, output ZERO is high. Because ofthe inverter, the final output ZERQis a~tive-lo~,Therefore, wheil th~ SUfil is zero, ZERO isnegativ7 true. gate need to be removed.These are Some of the bubbles .at the. inp~tofthebub.bled code word where '1' is J>rese11t, specifically S7 , S5 , S4 , S2 and S0 • AND Register Assertion-level logic diagr.im ,showing the detectio., of zero and mir:ms accumulator contents What isne~ative l?gic? What is meant assertion-level logic? 2~5 INTRODUCTION TO HOL In this section, we introduce an interesting development in the field of hardware design. This is textual description of a digital circuit. Though we have already described hardware, can there be a language which Digital Principles and Applications is more crisp and more importantly, machine-readable? The advantage of course, is to be able to (i) describe a large complex design requiring hundreds of logic gates in a convenient manner, in a smaller space, (ii) use software test-bench to detect functional error, if any, and correct it (called simulation) and finally, (iii) get hardware implementation details (called synthesis). Hardware Description Language, more popular with its acronym HDL is an answer for that. Currently, there are two widely used HDLs-Verilog and VHDL (Very high speed integrated circuit Hardware Description Language). Verilog is considered simpler of the two and is more popular. However, both share lot of common features and it is not too difficult to switch from one to the other. In this book, we'll deal with Verilog and shall discuss it over a span of number of chapters by introducing features relevant to that chapter. We expect by the time you finish Chapter 11, you'll have reasonable knowledge about HDL to deal with any digital logic design problem. We discuss target hardware devices on which HDL code can be directly exported in Section 13.6 of Chapter 13. Verilog HDl Verilog as a hardware descriptionlanguage has a small history. Introduced in 1980, primarily as a simulation and verification tool by Gateway Design Automation, it was later acquired by Cadence Data Systems. Put to public domain in 1990, it gained popularity and is now controlled by a group of companies and universities, called Open Verilog International. The reader with an exposure to any programming language like C will find it relatively easier to learn Verilog or any HDL. Describing Input/Output In any digital circuit, we find there are a set of inputs and a set of outputs. Often termed as ports, the relationship between these input and outputs are explained within the digital circuit. To design any circuit that has say, three inputs a, b, c and two outputs say, x, y as shown in Fig. 2.37 the corresponding Verilog code can be written as shown next. module testckt(x,y,a,b,c); //module name with port list a X b testckt input a,b,c; //defines input ports output x,y; //defines output ports //module body begins next describing logic relation y C //module body ends endmodule Input/output definition in Verilog HDL for logic circuit described within black-box testckt Note that, module and endmodule written in bold are keywords for Verilog. A module describes a design entity with a name or identifier selected by user (here, testckt) followed by input output port list. This entity if used by another then arguments (i.e. ports) are to be passed in the same order as it appears here. The symbol '//' is used to put comments and improve readability for a human but not used by the machine, i.e. compiler. The module body describes the logic within the black box which acts on the inputs a, b, c and generates outputx,y. Observe, where semicolon';' is used and where not to end a statement, e.g. endmodule in above code does not end with semicolon. Digital Logic Writing Module Body There are three different models of writing module body in Verilog HDL. Each one has its own advantage and suited for certain kind of design. We start with structural model by example of two-input OR gate described in Fig. 2.4a. module or_gate {A, B, Y); input A, B; / / defines two input port output Y; / / defines one output port o:r (Y,A,B); te declaration with predefined keyword logic OR, is optional user defi.ned encbnodule Verilog supports predefined gate level primitives such as and, or, not, nand, nor, xor, xnor etc. The syntax followed above can be extended to other gates and for 4 input OR gate itis as given next, or (output, input 1, input 2, input 3, input 4) For NOT gate, not (output, input) Note that, Verilog can take up to 12 inputs for logic gates. Comments when extends to next line is written within /* ..... */. Identifiers in Verilog are case sensitive, begin with a letter or underscore and can be of any length. Let us now look at description of a logic circuit shown in Fig. 2.17 a that has 4 inputs and 1 output. The inputs are fed to two 2-input AND gate. AND gate outputs are fed to a 2-input OR gate to generate final output. The verilog code for this is given below. Note that, we define two intermediate variables and_opl and and_op2 representing two AND gate outputs through keyword wire. Wire represents a physical wire in a circuit. module fig2_24a(A,B,C,D,Y); input A,B,C,D; output Y; wire and_opl, and_op2;· and gl(arid_opl,A, and g2(arid_op2,C,D); // g2 represents lower A.ND or g3(Y,and_opl,and_op2); // g3 represents the OR gate encbnodule One can see that structural model tries to replicate graphical layout design of a logic circuit. It does not matter if or statement in above example is written before and statements. This is as if one draws or collllects the OR gate first on a design board and then the AND gates. testckt X y Consider, the black box testckt of Fig. 2.38 has following logic circuit in it. Give Verilog structural code for the same. Logic circuit for Example 2.16 Digital Principles and Applications Solution The code from the above discussion can be written as follows. /* internal connecr:1.ons, outputs of upper */ gate Preparation of Test Bench We shall discuss data flow model and behavioral model ofVerilog VHDL in subsequent chapters. But, before we wind up this chapter let us see how to prepare a test bench in Verilog to simulate a digital circuit. For those of you with no programming background, this may appear little difficult. We could have postponed this discussion to a later chapter, but this gives you a feel of how simulation works or how a circuit you design can be tested. More clarity is assured as you go through discussions of subsequent chapters. We take up the example of simulating a simple OR gate (Fig. 2.4a) for which Verilog code is already described. The test bench, creates an input in the form of a timing waveform and passes this to OR gate module through a function or procedural can (passing arguments in proper order). To generate timing waveform we use time delay available in Verilog in the form of #n where n denotes a number in decimal that gives delay in nanosecond. Input values to a variable can be provided through syntax m'tn where m represents number of digits, t represents type of number and n represents value to be provided. The test bench used here generates all possible combinations of two inputs AB as 00,01,10 and 11 but at an interval of20 ns. Note that, we have provided a 20 ns gate delay with or statement by #(20). All practical logic circuit comes with finite gate delay, i.e. output changes according to input after certain time. To change the gate delay to 10 ns we should write #(10) in or statement. The keyword reg is used to hold value of a data object in a procedural assignment. The keyword initial ensures sequential execution of codes following it, but once. We'll learn another keyword always in later chapter, which too is used for sequential execution but for infinite time. a name of 20 ns AB=Ol Digital Logic endmodul.e module·or_gate input A,B; output ox,- # (20) gl (x,A, output E!ndmodule Execution of above Verilog code generates following timing diagram. One can see that input AB, given by testor.A and testor.B (testor is module name of the test bench) is taking value 00,01,10,11 as expected and retain them for 20 ns. Output of OR gate, testor.x changes according to input but after a delay of20 ns. For first 20 ns, OR gate output is unknown as it needs 20ns (gate delay) to respond to first appearance of input logic at A,B. Note that Verilog, in general offers four logic values in simulation 0, 1, unknown (or x) and high impedance (or z). Unknown value is exhibited when input is ambiguous and high impedance is shown when a wire by mistake is left unconnected or the circuit is following tri-state logic (Chapter 14, Section 6). !Ons i ! I testor.x ~ocAI testor.B I ! l ilOns l l I , 11201 ns1 I ! 1'30ns I I l 140ns i / / l \ f l ! j50ns! l l i i60ns ! l f l !70ns! ! ! l I Verilog simulation of 2 input OR gate with 20ns gate delay Write the statements between begin and end of a test bench for circuit described in Example 2.16 with 50 ns holding time of each input combination. Solution . Sincethe circuit hast~e~)nputs statements would look like as follows: 11eed 23 = 8 differe11t con1binati0lls .ofinputs.iThus ~e Digital Principles and Applications Delay .of ns ABC is assigned 011 II Delay of 50 ns bO; ABC is assigned 100 II Delay of 50 ns 'bl; ABC is assigned 101 II Delay of 50 ns 1 bO; II ABC is assigned 110 of ns II ABC is .assigned 111 terminates after 400 ns 'bl; bl; ;b=l' bO; IF50 a=l' bl; b=l' bO; l50 end The following timing diagram is generated by simulation of Verilog code for 2-input, 1-output device where A and B are input and x is output. Can you (i) estimate the gate delay and (ii) identify the logic? - i 0ns I 1 testand.x testand.A testand.B I I 11qns I 12on~ I i39n~ I [40ns ' 1sqn~ ' ' I \ I i6?n~ I I 170ns l I I I I Verilog simulation for Example 2.18 Solution (i) The unknown value of the output is approximately half of l Ons time scale. Hence, gate delay is 5ns. · (ii). Output goes HIGH when both the inputs go high after a delay of 5 ns. Hence, the logic underlying is AND. PROBLEM SOLVING WITH MULTIPLE METHODS Realize Y = AB + C using only one type of gate. Solution The functioilto berealized involves AND, NOT and OR operations. We can realize this expression using universal logic gate. AND from NOR gate r ------------ I I :A.B ,n Method-4, OR from NOR gate r-------------. we realize it using NOR g~te. We realize in~ividual logfo operations AA:eANI>,NOTandORasdepictedfoFig. 2.2LThe solutionis given in Fig. 2.41. ID Method-'.!, . werealize it using NANO gate. We reali;ze individual logic 9perations like AND,NOTandOR as depicted in Fig; 2.27. The solution is given in Fig. 2.42. I I f . '.' ·. -·· - - - · j :AB+C :c -------I NOT.frQlllNOR gate Realization of Y = AB + C using only NOR gate I ! Digital Logic Thus, two NOR operations X ==.X'Jrequiretwo.NOilgat~s,. ~ndfour inyersigns A,B,Clj!lllnal causes something to happen when low, it is drawn with a bubble; this is an activelow ·signal. Ifa signal causes something to happen when high, it is drawn without a bubble; this is an active-high signal. Combinational Logic Circuits + + + + + + Demonstrate the ability to use basic Boolean laws. Use the sum-of-products method to design a logic circuit based on a design truth table. Be able to make Karnaugh maps and Entered variable maps and use them to simplify Boolean expressions. Use the product-of-sums method to design a logic circuit based on a design truth table. Use Quine-McClusky tabular method for logic simplification Analyze hazards in logic circuit and provide solution for them. This chapter discusses Boolean algebra and several simplification techniques. After learning the laws and theorems of Boolean algebra, you can rearrange Boolean equations to arrive at simpler logic circuits. An alternative method of simplification is based on the Kamaugh map. In this approach, geometric rather than algebraic techniques are used to simplify logic circuits. Quine-McClusky tabular method provides a more systematic reduction technique, which is preferred when a large number of variables are in consideration. There are two fundamental approaches in logic design: the sum-of-products method and the productof-sums method. Either method produces a logic circuit corresponding to a given truth table. The sum-ofproducts solution results in an AND-OR or NAND-NAND network, while the product-of-sums solution results in an OR-AND or NOR-NOR network. Either can be used, although a designer usually selects the simpler circuit because it costs less and is more reliable. A practical logic circuit can show hazard due to finite propagation delay involved in each logic gate. This gives glitches or shows multiple transitions at the output. This chapter discusses different types of hazards and ways to prevent them. Combinational Logic Circuits 3.1 BOOLEAN LAWS ANO THEOREMS You should know enough Boolean algebra to make obvious simplifications. What follows is a discussion of the basic laws and theorems of Boolean algebra. Some of them will look familiar from ordinary algebra but others will be distinctly new. Basic laws The commutative laws are A+B=B+A AB =BA (3.1) (3.2) These two equations indicate that the order of a logical operation is unimportant because the same answer is arrived at either way. As far as logic circuits are concerned. Figure 3.la shows how to visualize Eq. (,.1). All it amounts to is realizing that the inputs to an OR gate can be transposed without changing the output. Likewise, Fig. 3.lb is a graphical equivalent for Eq. (3.2). The associative laws are A + (B + C) A(BC) = (A + B) + C = (AB)C ;=[)-r (3.3) (3.4) !=f:>-r (a) ~=[)-y !=[)-r (b) ;~y ~=e>-=D-y (c) A=C>;=[)- y B B~Y C C (d) B A y C (e) Commutative, associative, and distributive laws Digital Principles and Applications These laws show that the order of combining variables has no effect on the final answer. In terms oflogic circuits, Fig. 3.lc illustrates Eq. (3.3), while Fig. 3.ld represents Eq. (3.4). The distributive law is A(B + C) = AB + AC (3.5) This law is easy to remember because it is identical to ordinary algebra. Figure 3 .1 e shows the corresponding logic equivalence. The distributive law gives you a hint about the value of Boolean algebra. If you can rearrange a Boolean expression, the corresponding logic circuit may be simpler. The first five laws present no difficulties because they are identical to ordinary algebra. You can use these laws to simplify complicated Boolean expressions and arrive at simpler logic circuits. But before you begin, you have to learn other Boolean laws and theorems. OR Operations The next four Boolean relations are about OR operations. Here is the first: A+O=A (3.6) This says that a variable ORed with Oequals the variable. If you think about it, makes perfect sense. When A is 0, O+O =O And when A is 1, 1+0 = 1 In either case, Eq. (3.6) is true. Another Boolean relation is A+A =A Again, you can see right through this by substituting the two possible values of A. First when A (3.7) gives (3.7) = 0, Eq. O+O =O which is true. Next, A = I results in 1+ 1 = 1 which is also true because 1 ORed with 1 produces 1. Therefore, any variable ORed with itself equals the variable. Another Boolean rule worth knowing is A+ I= 1 (3.8) Why is this valid? When A= 0, Eq. (3.8) gives 0+1 =1 1+ 1 =1 which is true. Also. A = 1 gives This is correct because the plus sign implies OR addition, not ordinary addition. In summary, Eq. (3.8) says this, if one input to an OR gate is high, the output is high no matter what the other input. Combinational Logic Circuits Finally, we have A+ A= I You should see this in a flash. If A is 0, A is 1 and the equation is true. Conversely, if A is 1, the equation still agrees. In short, a variable ORed with its complement always equals I. (3.9) A is O and AND Operations Here are three AND relations A· 1 =A A ·A =A A ·O =O (3.10) (3.11) (3.12) When A is 0, all the foregoing are true. Likewise, when A is 1, each is true. Therefore, the three equations are valid and can be used to simplify Boolean equations. One more AND formula is A· A =O (3.13) This one is easy to understand because you get either 0·l =0 l· 0 =0 or for the two possible values of A. In words, Eq. (3.13) indicates that a variable ANDed with its complement always equals zero. Double Inversion and De Morgan's Theorems The double-inversion nde is A =A (3.14) which shows that the double complement of a variable equals the variable. Finally, there are the De Morgan theorems discussed in Chapter 2: (3.15) A+B =AB (3.16) AB= A +B You already know how important these are. The first says a NOR gate and a bubbled AND gate are equivalent. The second says a NAND gate and a bubbled OR gate are equivalent. Duality Theorem The duality theorem is one of those elegant theorems proved in advanced mathematics. We will state the theorem without proof. Here is what the duality theorem says. Starting with a Boolean relation, you can derive another Boolean relation by 1. Changing each OR sign to an AND sign 2. Changing each AND sign to an OR sign 3. Complementing any O or 1 appearing in the expression Digital Principles and Applications For instance, Eq. (3.6) says that A+O=A The dual relation is A· 1 =A This dual property is obtained by changing the OR sign to an AND sign, and by complementing the Oto get a 1. The duality theorem is useful because it sometimes produces a new Boolean relation. For example, Eq. (3 .5) states that A(B+C) =AB+AC By changing each OR and AND operation, we get the dual relation A + BC = (A + B)(A + C) (3.17) This is new, not previously discussed. (If you want to prove it, construct the truth table for each side of the equation. The truth tables will be identical, which means the Boolean relation is true.) Covering and Combination The covering rule, where one term covers the condition of the other term so that the other term becomes redundant, can be represented in dual form as A +AB =A A(A +B) =A and (3.18) (3.19) The above can be easily proved from basic laws because, and The combining rules are, A + AB = A · I +AB= A(I + B) = A · I = A A (A + B) = A · A +AB= A + AB = A AB+AB =A (A + B)(A + B) = A and in its dual form Eq. (3.20) can easily be proved as B + B (3.20) (3.21) =1 Expanding left hand side ofEq. (3.21) A·A+A·B+A·B+B·B =A+A(B+B)+O = A + A · 1 = A + A = A = right hand side Consensus Theorem The consensus theorem finds a redundant term which is a consensus of two other terms. The idea is that if the consensus term is true, then any of the other two terms is true and thus it becomes redundant. This can be expressed in dual form as AB+ AC+ BC =AB+ AC (A + B)(A + C) (B + C) = (A + B)(A + C) (3.22) (3.23) In the first expression, BC is the consensus term and thus redundant. This is because if BC= 1, then both B = 1 and C = 1 and any of the other two terms AB or AC must be one as either A = 1 or A = 1. Similarly, Combinational Logic Circuits in the second expression, (B + C) is the consensus term and if this term is O then both B = 0 and C = 0. This makes one of the other two sum terms Oas either A= 0 or A= 0. For future reference, here are some Boolean relations and their duals: A+B=B+A AB=BA A+ (B + C) =(A+ B) + C A(BC) = (AB)C A(B + C) =AB+ AC A +BC= (A+ B)(A + C) A+O=A A·l=A A+I =1 A-0=0 A+A=A A·A=A A+A =1 A· A =0 A=A A=A A+ B= AB A+AB=A A+AB=A+B AB= A+B A(A +B) =A A(A +B)=AB AB+AB =A (A+ B) (A+ AB+ AC+ BC= AB+ AC (A+ B) B) = A (A + C) (B + C) =(A+ B) (A Prove that, A(A' + C) (A'B + C) (A'BC + C') = 0 Solution LHS =(AA'+AC)(A'B+C)(A'BC+C') =AC(A'B + C) (A'BC+ C') =(AC· A'B +AC· C) (A'BC+ C') =AC(A'BC+ C') =AC A'BC+AC· C' =O=RHS : distributive law : since, XX' =0 : distributive law : since,){}{' = 0 : distributive law : since, .-IT' = 0 Simplify, Y= (A+ B) (A'(B' + C'))' + A'(B + C) Solution Y =(A+ B) ((A+ (B' + C')') +A'(B + C) . = (A+ JJ).(A +BC)+ A'(B + C) =(AA +ABC+ AB+ BBC)+ A'(B + C) =(A +AB +ABC+BC)+A'(B + C) =A(l + B +BC)+ BC+ A'(B + C) + BC +A 1(B + C) (A+ A'(B + C)) +BC =A +B + C+BC =A+ B + C(l + B) =A+B+C : De Morgan's theorem : De Morgan's theorem + C) Digital Principles and Applications A logic clip is a device that you can attach to a 14- or 16-pin DIP. This troubleshooting tool contains 16 light-emitting diodes (LEDs) that monitor the state of the pins. When a pin voltage is high, the corresponding LED lights up. If the pin voltage is low, the LED is dark. Suppose you have built the circuitofFig. 3.2a, but it doesn't work correctly. When you connect a logic clip to the 7408, you get the readings ofFig. 3.2b (a black circle means an LED is off, and a white one means it's on). When you connect the clip to the 7432, you get the indications of Fig. 3.2c. Which of the gates is faulty? A B y A B (a) e1 02 e3 04 05 06 97 140 13 0 120 11 0 100 90 80 (b) • =Off 0 =On fill 92 lll3 94 05 06 e7 140 130 120 110 100 90 80 (c) Solution When you use a logic clip, all you have to do is look at the inputs and output to isolate a faulty gate. For instance, Fig. 3.2b applies to a 7408 (quad 2-input AND gate). The First AND gate (pins l to 3) is all right because PinJ-,-low Pin2-high Pin3-low A 2-input AND gate is supposed to have a low output if any input is low. The second AND gate (pins 4 to 6) is defective, Why? Because Pfa4-high Pin5-high Pin6--low Something is wrong with this AND gate because it produces a low output even. though both inputs are high. If you check Fig. 3.2c (the 7432), all OR gates are normal. For instance, the first OR gate (pins! to3) is all right because it produces a low output when the 2 inputs are low. The second OR gate (pins 4 to 6) is working. correctly since it produces a high output when I input is high. L All the rules for Boolean algebra are exactly the same as for ordinary algebra. (Tor F) 2. Expand using the distributive law: Y = A(B + C). 3. Simplify: Y= AQ+ AQ. Combinational Logic Circuits 3.2 SUM-Of-PRODUCTS METHOD Figure 3 .3 shows the four possible ways to AND two input signals that are in complemented and uncomplemented form. These outputs are calledfimdamental products. Table 3.1 lists each fundamental product next to the input conditions producing a high output. For Fundamental Products for Two instance, AB is high when A and B are low; AB Inputs is high when A is low and Bis high; and so on. The fundamental products are also called minterms. Fundamental Product B A Products A' B', A'B, AB', AB are represented by 0 0 AB m0, mi, m2 , and m3 respectively. The suffix i ofmi AB 0 comes from decimal equivalent of binary values 0 AB (Table 3.1) that makes corresponding product term high. AB i =[]-AB ~ =[]-As ~ =[]-As ~ =[]-AB ~ 00 ~ ~ ANDing two variables and their complements The idea of fundamental products applies to three or more input variables. For example, assume three input variables: A, B, C and their complements. There are eight ways to AND three input variables and their complements resulting in fundamental products of ABC, ABC, ABC, ABC, ABC, AB c, ABC, ABC j~ C (a) Examples of ANDing three variables and their complements The above three variable minterms can alternatively be represented by mo, m1, 1112, 1113, 1114, 1115, 1116, and 111 7 respectively. Note that, for n variable problem there can be 211 number of minterms. Figure 3.4a shows the first fundamental product, Fig. 3.4b the second, and Fig. 3.4c the third. (For practice, draw the gates for the remaining fundamental products.) for twice variable case. Table 3.2 summarizes the fundamental products by listing each one next to the input condition that results in a high output. For instance, when A = 1, B = 0 and C = 0, the fundamental product results in an output of Y= ABC= 1 · 0 · 0 = 1 Fundamental Products for Three Inputs 0 0 0 1 1 1 I B Fundamental Products 0 ABC ABC ABC ABC ABC ABC ABC ABC 0 1 0 1 0 0 0 1 Digital Principles and Applications Sum-of-Products Equation Here is how to get the sum-of-products solution, given a truth table like Table 3.3. What you have to do is locate each output 1 in the truth table and write down the fundamental product. For instance, the first output 1 appears for an input of A = 0, B = 1, and C = 1. The corresponding fundamental product is ABC. The next output 1 appears for A = 1, B = 0, and C = 1. The corresponding fi.mdamental product is ABC. Continuing like this, you can identify all the fundamental products, as shown in Table 3.4. To get the sum-of-products equation, all you have to do is OR the fundamental products of Table 3.4: Y =ABC+ ABC+ ABC+ ABC (3.24) Alternate representation of Table 3.3, Y = F(A, B, C) = I: m (3, 5, 6, 7) where 'I:' symbolizes summation or logical OR operation that is performed on corresponding mintem1s and Y = F (A, B, C) means Y is a function of three Boolean variables A, B and C. This kind of representation of a truth table is also known as canonical sum.form. Design Truth Table A B C y 0 0 0 0 1 1 I I 0 0 1 1 0 0 1 1 0 1 0 0 0 0 1 0 1 1 1 1 0 l 0 1 Fundamental Products for Table 3.3 A B C y 0 0 0 0 0 0 1 0 1 0 l 0 0 l 0 l l 1 1 logic Circuit After you have a sum-of-products equation, you can derive the co1Tesponding logic circuit by drawing an AND-OR network, or what amounts to the same thing, a NAND-NAND network. In Eq. (3.24) each product is the output of a 3-input AND gate. Furthermore, the logical sum Yis the output of a4-input OR gate. Therefore, we can draw the logic circuit as shown in Fig. 3.5. This AND-OR circuit is one solution to the design problem that we started with. In other words, the AND-OR circuit ofFig. 3.5 has the truth table given by Table 3.3. We cannot build the circuit of Fig. 3.5 because a 4-input OR gate is not available as a TTL chip (a synonym for integrated circuit). But a 4-input NAND gate is. Figure 3.6 shows the logic circuit as a NAND-NAND circuit with TTL pin numbers. Also notice how the inputs come from a bus, a 1-, ABC 0 I-, ABC 1-, ABC 1 -,ABC 0 l I 0 I I 0 0 A B C ABC A B C ABC y A B ABC c A B C ABC AND-OR solution Combinational Logic Circuits group of wires carrying logic signals. In Fig. 3.6, the bus has six wires with logic signals A, B, C, and their complements. Microcomputers are bus-organized, meaning that the input and output signals of the logic circuits are connected to buses. Suppose a three-valuable trnth table has a high output for these input conditions: 000, 010, 100, and 110. What is the AABBCC sum-of-products circuit? Solution Here are the fundamental products: ooo: ABC 010: 100: ABC Alic 110 :ABC y When you OR these products, you get Y=ABC +ABC +ABC +ABC The circuit of Fig. 3.6 will work if we reconnect .the input lines to the bus as follows: 2 13 A :pins. I and 3 12 ----- B : .pins 2 and 10 C : pinsB, 5,11, and 13 o--~ 7410 A : pins 9 and! Combinational logic circuit B :pins4and2 Simplify the Boolean equation in Example 3.4 and describe the logic circuit. Solution The Boolean equation is Since y =ABC+ ABC+ ABC+ ABC C is common to each term, factor as follows: Y=(AB +AB+AB + AB)C Again, factor to get f;::[A(B +B)+A(B+B)]C Now, simplify the foregoing as follows: or r= [A (I)+ A(l)Jc = (A Y=c + A)c This final equation means that you don't even need a logic circuit; All you need is a wire connecting input· C to output Y. The lesson is clear. The AND-OR (NAND-NAND) circuit you get with the sum-of-products method isnotnecessarily as simple as possible. With algebra, you often can factor and reduce the .sum-ofcproducts equation to arrive at a simpler Boolean equation, which means a simpler logic circuit. A simpler logic circuit is prefon-ed because. it usually costs less to build.and is more reliable. Digital Principles and Applications 4. How many fondamental products are there for two variables? How many for three variables? 5. The AND-OR or the NAND-NAND circuit obtained with the sum-of-products method is always the simplest possible circuit. (T or F) 3.3 TRUTH TABLE TO KARNAUGH MAP A Karnaugh map is a visual display of the fundamental products needed for a sum-of-products solution. For instance, here is how to convert Table 3.5 into its Karnaugh map. Begin y A B by drawing Fig. 3.7a. Note the variables and complements: 0 0 0 the vertical column has A followed by A, and the horizontal I 0 0 1 0 1 row has B followed by B. The first output 1 appears for A = 1 1 1 and B = 0. The fundamental product for this input condition is AB . Enter this fundamental product on the Karnaugh map as shown in Fig. 3.7b. This 1 represents the product AB because the 1 is in row A and column Ji. Similarly, Table 3.5 has an output 1 appearing for inputs of A= 1 and B = 1. The fundamental product is AB, which can be entered on the Karnaugh map as shown in Fig. 3.7c. The final step in drawing the Karnaugh map is to enter Os in the remaining spaces (see Fig. 3.7d). In terms of decimal equivalence each position of Karnaugh map can be drawn as shown in Fig. 3.7b. Note that, Table 3.5 can be written using minterms as Y= L m(2, 3) and Fig. 3.7e represents that. B B B A~ A (a) I2 3 (b) (c) B B :~ B A~ A (d) I1 1 (e) Constructing a Karnaugh map Three-Variable Maps Here is how to draw a Karnaugh map for Table 3.6 or for logic equation, Y = F(A, B, C) = Lm(2,6,7). First, draw the blank map of Fig. 3.8a. The vertical column is labeled AB, AB, AB, and AB. With this order, only one variable changes from complemented to uncomplemented form (or vice versa) as you move downward. In terms of decimal equivalence of each position the Karnaugh map is as shown in Fig. 3.8b. Note how mintenns in the equation gets mapped into corresponding positions in the map. A .... 0 0 0 0 1 l 1 l B 0 0 1 I 0 0 1 l C 0 l 0 1 0 l 0 1 y 0 0 1 0 0 0 1 I Combinational Logic Circuits Next, look for output ls in Table 3.6. Output ls appear for ABC inputs of 010, 110 and 111. The fundamental products for these input conditions are ABC, ABC, and ABC Enter 1s for these products on the Kamaugh map (Fig. 3.8b). The final step is to enter Os in the remaining spaces (Fig. 3.8c). c AB AB AB AB c C AB AB AB AB (a) c C 0 2 3 6 7 4 5 C AB AB AB AB AB AB AB AB (b) c C 0 0 0 0 0 (d) (c) Three-variable Karnaugh map Four-Variable Maps Many digital computers and systems process 4-bit numbers. For instance, some digital chips will work with nibbles like 0000, 0001, 0010, and so on. For this reason, logic circuits are often designed to handle four input variables (or their complements). This is why you must know how to draw a four-variable Kamaugh map. Here is an

example. Suppose you have a truth table like Table 3.7. Start by drawing a blank map like Fig. 3.9a. Notice the order. The vertical column is AB, AB, AB, and AB. The horizontal row is CD, CD, CD, and CD. In terms of decimal equivalence of each position the Kamaugh map is as shown in Fig. 3.9b. In Table 3.7, you have output ls appearing for ABCD inputs of 0001, OllO, 0111, and 1110. The fundamental products for these input conditions are ABCD,ABCD,ABCD, and ABCD. After entering ls on the Karnaugh map, you have Fig. 3.9c. The final step of filling in Os results in the complete map of Fig. 3.9d. A B C D y 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 I I I 0 0 0 0 0 0 I 1 0 0 1 I. 0 0 0 I 0 0 I 0 0 0 0 J 1 0 0 0 1 0 1 0 1 0 1 0 6. WhatisaKarµaugh map? 7. How many entries are there on a four-variable Kamaugh map? Entered Variable Map As the name suggests, in entered variable map one of the input variable is placed inside Kamaugh map. This is done separately noting how it is related with output. This reduces the Karnaugh map size by one degree, Digital Principles and Applications ci5 CD G7J c:15 CD CD CD c:15 CD CD CD CD AB AB AB AB AB AB AB 0 3 2 4 5 7 6 12 13 15 14 8 9 11 10 AB AB AB AB 0 2 4 5 7 6 12 13 15 14 8 9 11 10 (b) (a) 3 cl5 CD CD CD 0 0 AB AB AB AB 0 0 0 0 0 0 0 0 0 1 0 (d) (c) Constructing a four-variable Karnaugh map i.e. a three variable problem that requires 2 3 = 8 locations in Karnaugh map will require i 3 -ll = 4 locations in entered variable map. This technique is particularly useful for mapping problems with more than four input variables. We illustrate the technique by taking a three variable example, truth table of which is shown in Table 3.6. Let's choose C as map entered variable and see how output Y varies with C for different combinations of other two variables A and B. Fig. 3.1 Oa shows the relation drawn from Table 3.6. For AB= 00 we find Y = 0 and is not dependent on C. For AB= 01 we find Y is complement of C thus we can write Y = C'. Similarly, for AB= 10, Y = 0 and for AB= 11, Y= 1. The corresponding entered variable map is shown in Fig. 3.10b. If we choose A as map entered variable we have table shown in Fig. 3 .1 Oc showing relation with Y for various combinations of BC; the corresponding entered variable map is shown in Fig. 3.10d. A B Y 0 0 0 c o 0 B B 7i~l' A IO 1 C Y 0 0 0 B~ 0 BI 0 0 C I A 0 I (a) c B (b) A (d) (c) Entered variable map of truth table shown in Table 3.6 3.4 PAIRS, QUADS, AND OCTETS Look at Fig. 3.lla. The map contains a pair of ls that are horizontally adjacent (next to each other). The first l represents the product AB CD; the second l stands for the product ABC I5. As we move from the first 1 to the second 1, only one variable goes from uncomplemented to complemented form (D to D); the other variables don't change form (A, Band C remain uncomplemented). Whenever this happens, you can eliminate the variable that changes form. cl5 AB AB AB AB CD CD CD 0 0 0 0 0 0 0 0 0 0 0 0 (a) 0 0 cl5 ~1:l AB 0 AB o CD CD CD 0 0 0 0 0 0 0 0 C[J) 0 (b) Horizontally adjacent 1 s 0 Combinational Circuits Proof The sum-of-products equation corresponding to Fig. 3.1 la is Y=ABCD + ABCD which factors into y = ABC(D + D) Since D is ORed with its complement, the equation simplifies to Y=ABC In general, a pair of horizontally adjacent Is like those of Fig. 3.11 a means the sum-of-products equation will have a variable and a complement that drop out as shown above. For easy identification, we will encircle two adjacent Is as shown in Fig. 3 .11 b. Two adjacent 1s such as these are called a pair. In this way, we can tell at a glance that one variable and its complement will drop out of the c01Tesponding Boolean equation. In other words, an encircled pair of ls like those of Fig. 3.1 lb no longer stand for the ORing of two separate products, ABCD and ABCD. Rather, the encircled pair is visualized as representing a single reduced product ABC. Here is another example. Figure 3.12a shows a pair of 1s that are vertically adjacent. These ls correspond to ABC D and AB CD. Notice that only one variable changes from uncomplemented to complemented form (B to B). Therefore, B and B can be factored and eliminated algebraically, leaving a reduced product of ACD. CD CD CD CD AB 0 0 0 0 AB 0 0 AB 0 AB 0 0 0 0 AB 0 0 0 0 AB 0 0 0 ~. (a) cB CD CD CD CD AB 0 0 AB 0 0 ~ CD CD c75 CD CD CD CD 0 AB 0 0 0 0 AB AB 0 0 AB 0 0 0 0 () 0 AB 0 0 0 0 0 0 AB 0 0 cc=J) (b) (c) () A~~ AB I 0 0 V method. Since 1 + C, we need a separate group for AB = 00 as C is not explained by other two groups. We use C embedded in 1 to make other two groups bigger and reduce the number of literals, and thus minimize the expression. - - - - - - - - ~ IA\ , Combinational Logic Circuits y 1 0 1 l 0 l l 0 0 J l 0 l 0 0 BC Y=AB+AC+BC Solution using Entered Variable Map In Method-4, we use QMalgorithmfor minimization. Fig. 3.41 shows prime implicants and essential prime implicants. The final solution is arrived at by combining essential prime implicants. Stage 1 Stage 2 ABC ABC 000 (0) 001 (l) 011 IOI (3) (5) '1 "' 0 00- (0, 1) 0-1 (1, 3) (1, 5) -01 3 A'B' '1 A'C " "' B'C All are essential 5 "' Y=A'B' + A'C+ B'C Prime implicants only from stage 2. Theyare: 00-(A'B'), 0-1 (A'C) and-01 (B'C) Solution using QM Algorithm Ev~ry Boolean equation has a dual form obtained by changing OR to AND, AND ~oOR, 0 to l,and 1 to O.J,Vith Boolean algebra you maybe able to simplify a Boolean equation, which implies a simplified logic circuit. Given a truth table, you can identify the fundamental products that produce output ls. By ORing these products, you get a sum-of-products equation for the truth table. A sum-of-products equation always results in an AND-OR circuit or its equivalent NAND-NAND circuit The Karnaugh method of simplifica.tion starts by converting a truth table intoaKarnaughmap.Ne}iit. You encircle all the octets, quads, and pain,. Tl:µ§ allo'W's you t9 W¥~e ;:1 simplified Boolean .equati9u an1:li!r thr? ughstrobe inpptifA ~ 0,th~top 7l4154is hos~naridBCDE directs datato.one.fth~ 15 outpt1~tth~tIC. If A = i,the bottorn: IC is chosen and depending valueofBCDEdata is directed to one ofthe15outputs this IC. on7 @ Digital Principles and Applications A B CD 20 21 22 23 1 18 2 3 4 5 DATA 6 _fl___fl_ t1 tz t3 t4 7 8 R T 19 74154 STROBE A Yo Yi Y2 ! Y3 Y4 74154 STROBE Y5 y6 Y1s BCDE !0 Y9 Yw 14 Y12 15 17 15 Y7 13 yil 16 Yo Y1 DATA 9 Ys 10 11 (a) BCDE Y13 Y14 Y15 D I Y16 Y17 15 Y31 DATA 74154 STROBE (b) :Alogicdtcuiftth one inp~t and many outputs is called a __ .. Forthe74154 demultiplexer, what must the logic levels ABCD be in order to steer the DATA inputsignalto outputline Y10? 5. IfAJJ X2 > X3. However, if the encoder is not enabled by Sor all the inputs are inactive the output AB= 00. Solution Figure 4.28b and .Fig, '4.28cshow the Karnaugh map for output A and B respectively. Note that, we have used a differentnotation.forinput variables lll these maps. Compare this with notations presented in previous chapters. You will find a variab.le with pri:rne is presented by O and if it is not primed is represented by 1. Then taking groups of Is we get the design equations as shown in the figure. The logic circuits for output A and B can be directly drawn from tllese equations. Intput Output s XI Xz X3 A B 0 X X X I 1 1 1 1 X X 0 0 0 1 X 0 0 I 0 0 0 1 1 0 0 I 0 1 0 S. I'i xx 2 3 00 01 00 0 0 0 0 11 0 0 10 0 0 01 A (a) 11 rn 0 0 SX1 XzX3 00 01 11 ·0 00 0 0 0 0 01 0 1 11 0 0 0 I 10 0 0 SXiX3 + s.1\Xz (b) l 0 0 B = SX1 + SXzX3 (c) Design of a priority encoder 12. What is an encoder? I3. What type of encoder is the TTL 74147? 4.7 EXCLUSIVE-OR GATES The exclusive-OR gate has a high output only when an odd number of inputs is high. Figure 4.29 shows how to build an exclusive-OR gate. The upper AND gate forms the product AB, while the lower one produces AB. Therefore, the output of the OR gate is A y Y=AB+AB Here is what happens for different inputs. When A and B B are low, both AND gates have low outputs; therefore, the final output Y is low. If A is low and B is high, the Exclusive-OR gate upper AND gate has a high output, so the OR gate has high output. Likewise, a high A and a low B result in a final output that is high. If both inputs are high, both AND gates have low outputs and the final output is low. Table 4.6 shows the truth table for a 2-input exclusive-OR gate. The output is high when A or Bis high, but not when both are high. This is why the circuit is known as an exclusive-OR gate. In other words, the output is a 1 only when the inputs are different. Digital Principles and Applications Exclusive-OR Truth Table A B y 0 0 0 0 I 1 1 0 I 1 0 ;==j[)-r Logic symbol for exclusive-OR gate Figure 4.30 shows the symbol for a 2-input exclusive-OR gate. Whenever you see this symbol, remember the action-the output is high if either input is high, but not when both are high. Stated another way, the inputs must be different to get a high output. Four Inputs Figure 4.3 la shows a pair of exclusive-OR gates driving an exclusive-OR gate. If all inputs (A to D) are low, the input gates have low outputs, so the final gate has a low output. If A to C are low and D is high, the upper gate has a low output, the lower gate has a high output, and the output gate has a high output. Ifwe continue analyzing the circuit operation for the remaining input possibilities, we can work out Table 4. 7. Here is an important prope1ty of this truth table. Each ABCD input with an odd number of ls produces an output I. For instance, the first ABCD entry to produce an output 1 is 000 I; it has an odd '-",::·· ·.. . _., able 4.7 · Comment Even Odd Odd Even Odd Even Even Odd Odd Even Even Odd Even Odd Odd Even A B y C D (a) !~y (b) Four-input exclusive OR gate 4-lnput Exclusive-OR Gate A B C D y 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 I 0 I 0 0 I 1 0 1 0 0 1 0 0 0 0 1 1 0 0 I 0 0 I 1 0 0 0 I 0 I 0 1 0 0 I I 0 0 I 0 1 I 0 Data-Processing Circuits number of 1s. The next ABCD entry to produce an output 1 is 001 O; again, an odd number of Is. An output 1 also occurs for these ABCD inputs: 0100, 0111, 1000, 1011,1101, and 1110, each having an odd number of ls. Figure 4.3 la illustrates the logic for a 4-input exclusive-OR gate. In this book, we will use the abbreviated symbol given in Fig. 4.31 b to represent a 4-input exclusive-OR gate. When you see this symbol, remember the action-the gate produces an output 1 when the ABCD input has an odd number of ls. Any Number of Inputs Using 2-input exclusive-OR gates as building blocks, you can produce exclusive-OR gates with any number of inputs. For example, Fig. 4.32a shows a pair of exclusive-OR gates. There are 3 inputs and l output. If you analyze this circuit, you will find it produces an output 1 only when the 3-bit input has an odd number of ls. Figure 4.32b shows an abbreviated symbol for a 3-input exclusive-OR gate. (a) (b) (c) (d) Exclusive-OR gate with several inputs As another example, Fig. 4.32c shows a circuit with 6 inputs and 1 output. Analysis of the circuit shows that it produces an output 1 only when the 6-bit input has an odd number of 1s. Figure 4.32d shows an abbreviated symbol for a 6-input exclusive-OR gate. In general, you can build an exclusive-OR gate with any number of inputs. Such a gate always produces an output I only when the n-bit input has an odd number of Is. 14. When is the output of an exclusive-OR gate high? 15. Draw the logic symbol for an exclusive-OR gate. 4~8 PARITY GENERATORS AND CHECKERS Even parity means an n-bit input has an even number of ls. For instance, 110011 has even parity because it contains four ls. Odd parity means an n-bit input has an odd number of ls. For example, 110001 has odd parity because it contains three ls. Principles and Applications Here are two more examples: 1111 0000 1111 0011 even parity 1111 0000 1111 0111 odd parity The first binary number has even parity because it contains ten ls; the second binary number has odd parity because it contains eleven 1s. Incidentally, longer binary numbers are much easier to read if they are split into nibbles, or groups of four, as done here. Parity Checker 10101 100100 01100 Exclusive-OR gates are ideal for checking the parity of a binary number because they produce an output 1 when the input has an odd number of 1s. Therefore, an even-parity input to an exclusive-OR gate produces a low output, while an odd-parity input produces a high output. For instance, Fig. 4.33 shows a 16-input exclusive-OR gate. A 16-bit number drives the input. The exclusive-OR gate produces an output 1 because the input has odd parity (an odd number of ls). If the 16-bit input changes to another value, the output becomes O for even-parity numbers and 1 for odd-parity numbers. Exclusive-OR gate with 16 inputs Parity Generation In a computer, a binary number may represent an instruction that tells the computer to add, subtract, and so on; or the binary number may represent data to be processed like a number, letter, etc. In either case, you sometimes will see an extra bit added to the original binary number to produce a new binary number with even or odd parity. For instance, Fig. 4.34 shows this 8-bit binary number: X1X6XsX4 X3X2X1 Xo Suppose this number equals OI 00 0001. Then, the munber has even parity, which means the exclusive-OR gate produces an output of 0. Because of the inverter, 8-bit number Xs= 1 and the final 9-bit output is 1 01000001. Notice that this has odd parity. Suppose we change the 8-bit input to O110 0001. Now, it has odd parity. In this case, the exclusive-OR gate produces an output 1. But the inve1ter produces a 0, so that the final 9-bit output is O O110 0001. Again, the final output has odd parity. The circuit given in Fig. 4.34 is called an odd-parity generator because it always produces a 9-bit output number witn odd parity. If the 8-bit input has even parity, a 1 comes I ~ X. 1 Instruct10n or data btts 9-bit number with odd parity Odd-parity generation Data-Processing Circuits out of the inverter to produce a final output with odd parity. On the other hand, if the 8-bit input has odd parity, a O comes out of the inverter, and the final 9-bit output again has odd parity. (To get an even-parity generator, delete the inverter.) Application What is the practical application of parity generation and checking? Because of transients, noise, and other disturbances, 1-bit errors sometimes occur when binary data is transmitted over telephon,' lines or other communication paths. One way to check for errors is to use an odd-parity generator at the iansmitting end and an odd-parity checker at the receiving end. If no 1-bit errors occur in transmission, the received data will have odd parity. But if one of the transmitted bits is changed by noise or any other disturbance, the received data will have even parity. For instance, suppose we want to send 0100 0011. With an odd-parity generator like Fig. 4.34, the data to be transmitted will be O O100 0011. This data can be sent over telephone lines to some destination. If no errors occur in transmission, the odd-parity checker at the receiving end will produce a high output, meaning the received numqer has odd parity. On the other hand, if a 1-bit error does creep into the transmitted data, the odd-parity checker will have a low output, indicating the received data is invalid. One final point should be made. Errors are rare to begin with. When they do occur, they are usually 1-bit e1rnrs. This is why the method described here catches almost all of the errors that occur in transmitted data. The 74180 Figure 4.35 shows the pinout diagram for a 74180, which is a TTL parity generator-checker. The input data bits are X 7 to X0 ; these bits may have even or odd parity. The even input (pin 3) and the odd input (pin 4) control the operation of the chip as shown in Table 4.8. The symbol I stands for summation. In the left input column of Table 4.8, I of H's (highs) refers to the parity of the input dataX7 toX0 . Depending on how you set up the values of the even and odd inputs, the I even and I odd outputs may be low or high. For instance, suppose even input is high and odd input is low. When the input data has even parity (the first entry of Table 4.8), the I even output is high and the I odd output is low. When the input data has odd parity, the I even output is low and the 1: odd output is high. 74180 Truth Table EVEN INPUT ODD INPUT Outputs Inputs 74180 I ofH's at Even Odd I even Iodd I: EVEN OUTPUT I:ODDOUTPUT GND Pinput diagram of 74180 H L L H L H H L L H Digital Principles and Applications lfyou change the control inputs, you change the operation. Assume that the even input is low and the odd input is high. When the input data has even parity, the 2: even output is low and the 2: odd output is high. When the input data has odd parity, the I even output is high and the I odd output is low. The 74180 can be used to detect even or odd parity. It can also be set up to generate even or odd parity. 1 2 8 9 +5V t1 3 p ODD INPUT EVEN INPUT 10lu li2113 74180 :[ODD OUTPUT 16 Using a 74180 to generate odd parity xa!!'p.e .1-:12 Show how to connect a 74180 to geJ1erate a 9-bit output with odd parity. Solution Figure 4.36 shows one solution. The ODD INPUT (pin 4) is connected to +5 V, and the EVEN INPUT (pin 3) is grounded. Suppose the input data X7 ... Xo has even parity. Then, the third entry of Table 4.8 tells us the .:E ODD OUTPUT (pin 6) is high. Therefore, the 9-bit number Xs ... Xo coming out of the circuit has odd parity. On the other hand, supposeX7 ••• Xo has odd parity. Then the fourth entry of Table 4.8 says that the L odd output is low. Again, the 9-bit number Xg ... Xi> coming out at the bottom of Fig. 4.36 has odd parity. The following conclusion may be drawn. Whether the input data has even or odd parity, the 9-bit number being generated in Fig. 4.36 always has odd parity. 16. What does it mean to say that an n-bit binary number has even parity? 17. Exclusive-OR gates are useful as parity generators. (Tor F) 4.9 MAGNITUDE COMPARATOR Magnitude comparator compares magnitude two n-bit binary numbers, say X and Yand activates one of these three outputs X = Y, X > Y and ,:r < Y. Figure 4.37a presents block diagram of such a comparator. Fig. 4.37b presents truth table when two I-bit numbers are compare Y): G =XY' (X < Y): L =X'Y (X= Y): E =X'Y' + XY = (XY' + X'Y)' = (G + L)' Data-Processing Circuits }-1 n~2 (X> Y) Xo n-bit comparator y•-1 11-2 Input X y (X= Y) (X< Y) Yo (a) Output X>Y X=Y X Y ifMSB of Xis higher (Gi = 1) than that of Y. If MSB is equal, given by E 1 = 1, then LSB of X and Y is checked and if found higher (G0 = I) the condition X > Y is fulfilled. Similar logic gives us the X < Y term. Thus for any two n-bit numbers X: X 11 _ 1 X,,_z .. .Xo and Y: Y11 _i Yn-2· . .Yo We can write, (X= Y) =En-i En-2···Eo (X> Y) = Gn-1 + En-I Gn-2 + ... +En-I En-2··· Ei Go (X < Y) = Ln-1 + En-ILn-2 + ... + En-i En-2··· E1Lo where E;, G; and L; represent for ith bit X; = Y; , X; > Y; and X; < Y; terms respectively. The block diagram ofIC 7485, which compares two 4-bit numbers is shown in Fig. 4.38a. This is a 16 pin IC and all the pin numbers are mentioned in this functional diagram. Note that the circuit has three additional inputs in the form of (X = lJin, (X> lJin and (X < lJin· What is the use of them? They are used when we need to connect more than one IC 7485 to compare numbers having more than 4-bits. But these inputs are oflower priority. They can decide the output only when 4-bit numbers fed to this IC are equal. For example, if X = 0100 and Y= 0011, (X> Y) 0 ut will be high and other outputs will be low irrespective of the value appearing at (X= lJin, (X> lJin and(X< lJin· When IC 7485 is not used in cascade we keep (X= Y)i11 = 1, (X> lJin =O and (X< lJin=O. Digital Principles and Applications X y ..-------, ..--------, X3X2X1Xo Y3 Yz Y1 Yo X3X2X1Xo ! ! !1 15131210 Vcc(16) GND(8) 5 1 1411 9 4 IC 7485 3 72 6 ! 15131210 ~X> Y) X-Bm (X~ ;~ 5 l 1411 9 4 IC 7485 3 72 6 15131210 5 Y3 Y2Y1 Yo l! ! +5v 1 1411 9 4 IC 7485 3 72 6 // (X> Y)out (X= Y)out (X < Y)out (X> Y)out (X = Y)out (X < Y)out (a) (b) (a) functional diagram of IC 7485, (b) 8-bit comparator from two 4-bit comparators Show how two IC 7485 can be used to compare magnitude of two 8-bit numbers. x. . . Y1. Solution Refer to Fig. 4.38h for iolution. The numbersto compare are ;Xo mtd Y: Y6: .. Yo. We nfed two IC 7485s each one comparing 4 bits. The most significant bits (suffix 7,6,5,4) are given higher;priority and tbe final output is tllken from that IC 7485 which compares them. 18. How many outputs a magnitude comparator generates? 19. How many IC 7485s ate needed to compare two 12-bit numbers? 4. to READ-ONl Y MEMORY A read-only mem01y (which is abbreviated ROM and rhymes with Mom) is an IC that can store thousands of binary numbers representing computer instructions and other fixed data. A good example of fixed data is the unchanging information in a mathematical table. Since the numerical data do not change, they can be stored in a ROM, included in a computer system, and used as a "look-up" table when needed. Some of the smaller RO Ms are also used to implement truth tables. In other words, we can use a ROM instead of sum-of-products circuit to generate any Boolean function. Diode ROM Diode ROM Suppose we want to build a circuit that stores the binary numbers shown in Table 4.9. To keep track of where the numbers are stored, we will assign addresses. For instance, we want to store 0111 at address 0, 1000 at address 1, 1011 at address 2, and so forth. Figure 4.39 shows one way to store the nibbles given in Table 4.9. When the switch is in position O (address 0), the upper row of diodes are conducting current (they act as closed Nibble 6 7 0111 1000 1011 llOO 0110 1001 0011 1110 Data-Processing Circuits switches). (See Chapter 14 for a discussion of diodes.) The output of the ROM is thus Y3 Y2 Y1 Yo = 0111 When the switch is moved to position I, the second row is activated and Y3 Y2 Y1 Yo = I 000 As you move the switch to the remaining positions or addresses, you get a Y3 the nibbles given in Table 4.9. ••• Yo output that matches 0 2 3 +5V 4 5 7 6 Diode ROM On-Chip Decoding Rather than switch-select the addresses as shown in Fig. 4.39, a manufacturer uses on-chip decoding. Figure 4.40 illustrates the idea. The 3-input pins (A, B, and C) supply the binary address of the stored number. Then, a l-of-8 decoder produces a high output to one of the diode rows. For instance, if ABC= 100 the l-of-8 decoder applies a high voltage to the ABC line, and the ROM output is Y3Y2Y1 Yo= 0110 Digital Principles and Applications If you change the binary address to ABC= 110 the ROM output changes to Y3 Y2 Y1 Yo= 0011 With on-chip decoding, n inputs can select 2 11 memory locations (stored numbers). For instance, we need 3 address lines to access 8 memory locations, 4 address lines for 16 memory locations, 8 address lines for 256 memory locations, and so on. Commercially Available ROMs A binary number is sometimes called a word. In a computer, binary numbers or words represent instructions, alphabet letters, decimal numbers, etc. The circuit given in Fig. 4.40 is a 32-bit ROM organized as 8 words A B C Alic On-chip decoding _ _ _ _ _ _ _ _ _ _ _ _ _o_a__ta_-P_ro_ce_ss_in_g_C_in_cu_it_s _ _ _ _ _ _ _ _ _ _ _ _ _ _ ~ with 4 bits at each address (an 8 x 4 ROM). The ROM given in Fig. 4.40 is for instructional purposes only because you would not build this circuit with discrete components. Instead, you would select a commercially available ROM. For instance, here are some TTL ROMs: 7488: 256 bits organized as 32 x 8 74187: 1024 bits organized as 256 x 4 74S370: 2048 bits organized as 512 x 4 As you can see, the 7488 can store 32 words of8 bits each, the 74187 can store 256 words of 4 bits each, and the 74S370 can store 512 words of 4 bits each. If you want to store bytes (words with 8 bits), then you can parallel the 4-bit ROMs. For example, two parallel 74187s can store 256 words of8 bits each. One way to change the stored numbers of a ROM is by adding or removing diodes. With discrete circuits, you would have to solder or unsolder diodes to change the stored nibbles. With integrated circuits, however, you can send a list of the data to be stored to an IC manufacturer, who then produces a mask (a photographic template of the circuit). This mask is used in the mass production of your ROMs. As a rule, ROMs are used only for large production runs (thousands or more) because of manufacturing costs. Generating Boolean Functions Because the AND gates of Fig. 4.40 produce all the fundamental products and the diodes OR some of these products, the ROM is generating four Boolean functions as follows: Y3 = ABC+ABC +ABC+ABC+ABC Y2 =ABC+ ABC+ ABC+ ABC Y1 =ABC+ ABC+ ABC+ ABC+ ABC Y0 =ABC+ ABC+ ABC+ ABC (4.1) (4.2) (4.3) (4.4) This means that you can use a ROM instead of a logic circuit to implement a truth table. For instance, suppose you start with a truth table like the one in Table 4.10. There are four outputs: Y3, Y2 , Y1, and Y0. A sum-of-products solution would lead to four AND-OR circuits, one for Eq. (4.1), a second for Eq. (4.2), and so on. The ROM solution is different. With a ROM you have to store the binary numbers of Table 4.9 (same as Table 4.10) at the indicated addresses. When this is done, the ROM given in Fig. 4.40 is equivalent to a sum-of-products circuit. In other words, you can use the ROM instead of an AND-OR circuit to generate the desired truth table. Truth Table A B C Y3 Y2 Y1 Yo 0 0 0 0 1 I I I 0 0 1 1 0 0 I 0 I 0 1 0 I 0 0 I 1 0 0 1 I 0 1 0 1 0 1 1 1 0 1 0 0 1 I I 0 1 0 1 I 1 1 0 1 0 I 0 Digital Principles and Applications Programmable ROMs A programmable ROM (PROM) allows the user instead of the manufacturer to store the data. An instrument called aPROMprogrammer stores the words by "burning in." Here is an example ofhow a PROM programmer works. Originally, all diodes are connected at the cross points. For instance, in Fig. 4.40 there would be a total of32 diodes (8 rows and 4 columns). Each of these diodes has afi1sible link (a small fuse). The PROM programmer sends destructively high currents through all diodes to be removed. In this way, only the desired diodes remain connected after programming a PROM. Programming like this is permanent because the data cannot be erased after it has been burned in. Here are some commercially available PROMs: 74Sl88: 256 bits organized as 32 x 8 74S287: ·1024 bits organized as 256 x 4 74S472: 4096bits organized as 512 x 8 PRO Ms such as these are useful for small production runs. For instance, if you are building only a few hundred units (or maybe even just one), you would choose a PROM rather than a ROM. Since PROMs are useful in many applications, manufacturers produce these chips in high volume. Furthermore, the PROM is a universal logic solution. Why? Because the AND gates generate all the fundamental products; the user can then OR these products as needed to generate any Boolean output. One disadvantage of PRO Ms is the limit on number of input variables; typically, PRO Ms have 8 inputs or less. Simplified Drawing of a PROM It is cumbersome to draw large PROMs as illustrated in Fig. 4.41, because of the large number of diodes. An alternative, streamlined drawing procedure for PROMs like the one in Fig. 4.40 is shown in Fig. 4.41. In this simplified drawing, the solid black A B C bullets indicate connections to the ANDgate inputs. Each bullet represents a fixed Programmable OR array connection that cannot be changed. Furthermore, each AND gate has 3 inputs, indicated by the bullet on .its input line. Similarly, each OR gate has 8 inputs, as indicated by the x 's on its input line, but each x is a fusible link that can. be · removed. Notice that the input side of Fig. 4.41 is a fixed AND array, meaning the inputs to the AND gates are not programmable in a PROM. On the other hand, the output side of the circuit is programmable because each connection at the input of Fixed AND array each OR gate is a fusible link. A fixed AND array and a programmable OR arY3 Y2 Y1 Yo ray are characteristic of all PRO Ms. To Streamlined drawing of PROM begin with, every AND-gate output is connected to every OR-gate input. Since Data-Processing Circuits the AND gates produce all eight possible combinations of the input variables A, Band C, it is possible to produce any Boolean function at the OR-gate outputs. Programmable OR array 1----11---1--,iE---1-ABC 1-----1e----,...__,..._...,._]BC Programming a PROM Generating a Boolean function at the output of a PROM is accomplished by fusing (melting) fusible links at the input to the OR gates in Fig. 4.41. For example, suppose we want to generate the function Yo = ABC. Simply fuse (melt) 7 of the AND-gate outputs connected to the Yo OR-gate input and leave the single AND-gate output ABC connected. A portion of Fig. 4.41 is shown in Fig. 4.43 with the proper fusible link remaining for Y0 . As a second example, suppose we want to generate the function Y1 = AB . We must include all terms containing AB, since ABC +ABC= AB(C + C)=AB 1----11---+---l'---+-ABC l----11---+---1'---t-ABC 1-----;,1E---l!E---ll---+-ABC 1---.;i1E---llt'---ll---+-ABC 1-----1e---_,._---1f---+-ABC 1----11----;tE---1'---J!:-- ABC Boolean function from PROM The two top fusible links must be included, while the remaining six are broken, as shown in Fig. 4.42. Continuing in this fashion, you can see that Y2 = A and Y3 =AB. Erasable PROMs The erasable PROM (EPRO.M) uses metal-oxide-semiconductor field-effect transistors (MOSFETs). Data is stored with an EPROM programmer. Later, data can be erased with ultraviolet light. The light passes through a quartz window in the IC package. When it strikes the chip, the ultraviolet light releases all stored charges. The effect is to wipe out the stored contents. In other words, the EPROM is ultraviolet-light-erasable and electrically reprogrammable. Here are some commercially available EPROMs: 2716: 16,384 bits organized as 2048 x 8 2732: 32,768 bits organized as 4096 x 8 The EPROM is useful in project development. With an EPROM, the designer can modify the contents until the stored data is perfect. When the design is finalized, the data can be burned into PROMs (small production runs) or sent to an IC manufacturer who produces ROMs (large production runs). 20. What is a ROM? 21. What does it mean to say that a particular ROMjs 22. What is a PROM? Digital Principles and Applications • 4.11 PROGRAMMABLE ARRAY LOGIC I Programmable array logic (PAL) is a programmable array of logic gates on a single chip. PALs are another design solution, similar to a sum-of-products solution, product-of-sums solution, and multiplexer logic. Programming a PAL A PAL is different from a PROM because it has a programmable AND array and a fixed OR array. For instance. Fig. 4.43 shows a PAL with 4 inputs and 4 outputs. The x's on the input side are fusible links, while the solid black bullets on the output side are fixed connections. With a PROM programmer, we can burn in the desired fundamental products, which are then ORed by the fixed output connections. A B C D Fixed OR array Programmable AND array Y3 ce::~ti;:f}I) Y2 Y1 Yo Structure of PAL Here is an example of how to program a PAL. Suppose we want to generate the following Boolean functions: Data-Processing Circuits Y3 = ABC D + A.BCD + ABCD + ABCD Y2 = ABCD +A.BCD+ ABCD Y, = ABC +ABC+ABC+ABC (4.5) (4.6) (4.7) Yo =ABCD (4.8) Start with Eq. (4.5). The first desired product is A.BCD. On the top input line of Fig. 4.44 we have to remove the first x, the fourth x, the fifth x, and the eighth x. Then the top AND gate has an output of ABCD. By removing xs on the next three input lines, we can make the top four AND gates produce the fundamental products ofEq. (4.5). The fixed OR connections on the output side imply that the first OR gate produces an output of 1:3 = ABCD + ABCD + ABCD + ABCD A C B D ti R-7 \7 v Fixed OR array i, Programmable AND array Example of programming a PAL Digital Principles and Applications Similarly, we can remove xs as needed to generate Y2, Y1, and Y0 . Figure 4.44 shows how the PAL looks after the necessary xs have been removed. If you examine this circuit, you will see that it produces the Y outputs given by Eqs. (4.5) to (4.8). Commerdaily Available PALs The PAL given in Fig. 4.43 is hypothetical. Commercially available PALs typically have more inputs. For instance, here is a sample of some TTL PALs available from National Semiconductor Corporation: 10H8; 10 input and 8 output AND-OR 16H2: 6 input and 2 output AND-OR 14L4: 14 input and 4 output AND-OR-INVERT For these chip numbers, H stands for active-high output and Lfor active-low output. The 10H8 and the 16H2 produce active-high outputs because they are AND-OR PALs. The 14L4, on the other hand, produces an active-low output because it is an AND-OR-INVERT circuit (one that has inverters at the final outputs). Unlike PROMs, PALs are not a universal logic solution. Why? Because only some of the fundamental products can be generated and ORed at the final outputs. Nevertheless, PALs have enough flexibility to produce all kinds of complicated logic functions. Furthermore, PALs have the advantage of 16 inputs compared to the typical limit of 8 inputs forPROMs. 23. What is a PAL? 24. A PAL has an AND array and an OR array. Which one is fixed and which is programmable? 4.12 PROGRAMMABLE LOGIC ARRAYS Programmable logic arrays (PLAs), along with ROMs and PALs, are included in the more general classification ofICs called programmable logic devices (PLDs). Figure 4.45 illustrates the basic operation of these three PLDs. In each case, the input signals are presented to an array of AND gates, while the outputs are taken from an array of OR gates. The input AND-gate array used in a PROM is fixed and cannot be altered, while the output OR-gate array isji,sible-linked, and can thus be programmed. The PAL is just the opposite: The output OR-gate array is fixed, while the input AND-gate array is fusible-linked and thus programmable. The PLA is much more versatile than the PROM or the PAL, since both its AND-gate array and its OR-gate array are fusible-linked and programmable. It is also more complicated to utilize since the number of fusible links are doubled. A PLA having 3 input variables (ABC) and 3 output variables (XYZ) is illustrated in Fig. 4.46. Eight AND gates are required to decode the 8 possible input states. In this case, there are three OR gates that can be used to generate logic functions at the output. Note that there could be additional OR gates at the output if desired. Programming the PLA is a two-step process that combines procedures use~ with the PROM and the PAL. As an example, suppose it is desired to use a PLA to recognize each of the 10 decimal digits represented in binary form and to correctly drive a 7-segment display. The 7-segment indicator was presented in Sec. 4.5. To begin with, the PLA must have 4 inputs, as shown in Fig. 4.47a. Four bits (ABCD) are required to represent the 10 decimal numbers (see Table 1.1). There must be 7 outputs (abcdefg), 1 output to drive each Data-Processing Circuits Input Fixed ,___ _ _ _ _ Fusible AND OR Array Array A B C Output Programmable OR array ,---------"--- PROM Input Fusible ,___ _ _ _ __ AND Array Fixed OR Array Output Fusible OR Array Output PAL Input Fusible 1--------1 AND Array Programmable AND array PLA X Y Z of the 7 segments of the indicator. Let's assume that our PLA is capable of driving the 7-segment indicator directly. (This is not always a valid assumption, and a buffer amplifier may be needed to supply the proper current for the indicator.) To begin with, all fusible links are good. The circuit in Fig. 4.47b shows the remaining links after programming. The input AND-gate array is programmed (fusible links are removed) such that each AND gate decodes one of the decimal numbers. Then, with the use of Fig. 4.47c, links are removed from the output OR-gate array such that the proper segments of the indicator are illuminated. For instance, when ABCD = LHLH, segments afgcd are illuminated to display the decimal number 5. You should take the time to examine the other nine digits to confirm proper operation. One final point. Many PLDs are programmable only at the factory. They must be ordered from the manufacturer with specific programming instructions. There are, however, PLDs that can be programmed by the user. These are said to be field-programmable, and the letter F is often used to indicate this fact. For instance, the Texas Instruments TIFPLA840 is a field-programmable PLA with 14 input variables, 32 AND gates, and 6 OR gates; it is described as a 14 x 32 x 6 FPLA. Wbatis aPLA'l How does a PLA differ from a PAL? 27. In Fig. 4.47, ABCD =LLHH. What segments are activated? Digital Principles and Applications a A b B C a .rjf C p L A d e/ f e l D g g d (a) A B C (c) D v· ~i ~·R7 0 I 2 3 4 5 6 7 8 9 a b c d e f g (b) 7-segment decoder using PLA 4.13 TROUBLESHOOTING WITH A LOGIC PROBE Chapter 3 introduced the logic clip, a device that connects to a 14 or 16-pin TC. The logic clip contains 16 LEDs that monitor the state of the pins. When a pin voltage is high, the corresponding LED lights up. When the pin voltage is low, the LED is dark. Figure 4.48 shows a logic probe, which is another troubleshooting tool you will find helpful in diagnosing faulty circuits. When you touch the probe tip to the output node as shown, the device lights up for a high state and goes dark for a low state. For instance, if either A or B, or both. arc low, then Yis high and the probe lights up. On the other hand, if A and B are both high, Y is low and the probe is dark. Data-Processing Circuits Among other things, the probe is useful for locating short circuits that occur in manufacturing. For example, during the stuffing and soldering of printed-circuit boards, an undesirable splash of solder may connect two adjacent traces (conducting lines). Known as a solder bridge, this kind of trouble can shortcircuit a node to the ground or to the supply voltage. The node is then stuck in a low or high state. The probe helps you to find short-circuited nodes because it stays in one state, no matter how the inputs are changing. Bright or dark Using a logic probe 4.14 HDI IMPLEMENTATION OF DATA PROCESSING CIRCUITS We start with hardware design of multiplexers using Verilog code. The data flow model provides a different use of keyword assign in the fom1 of = S ? A : B; This statement does following assignment. If, S = 1, X = A and if S = 0, X = B. One can use this statement assign X or the logic equation to realize a 2 to 1 multiplexer shown in Fig. 4.2(a) in one of the following ways. module mux2tol(A,D0,D1,Y); module mux2tol(A,DO,Dl,Y); input A,DO,Dl; /* Circuit shown input A,DO,Dl; /* Circuit shown in in Fig. 4.3(a) */ Fig. 4.3(a)*/ output Y; output Y; assign Y=(-A&D0) I (A&Dl); assign Y= A? Dl DO; /*Conditional endmodule assignment*/ endmodule The behavioral model can be used to describe the 2 to 1 multiplexers in following two different ways, one using if ... else statement and the other using case statement. The case evaluates an expression or a variable that can have multiple values each one corresponding to one statement inthe following block. Depending on value of the expression, one of those statements get executed. The behavioral model of2 to 1 multiplexer in both is given below: module mux2tol(A,DO,Dl,Y); input A,DO,Dl; /* Circuit shown in Fig. 4.3(a)*/ output Y; reg Y; (A or DO or Dl) if {A==l) Y=Dl; else Y=DO; endmodule module mux2tol(A,D0,Dl,Y); input A,DO,Dl; /* Circuit shown in Fig. 4.3(a) */ output Y; reg Y; (A or DO or Dl) case (A) 0 : Y=DO; 1 : Y=Dl; endcase endmodule Digital Principles and Applications Design a 4 to 1 multiplexer, shown in Fig. 4.l(c) using conditional assign and case statements. The codes are given next. We have used nested condition for assign statement. If AP 1, coµdition . D2) is evaluated. Then if B = 1, Y =. D3. And this is whatis given in l(c). Similarly, the other (B ? combinations of A and B are evaluated and Y is assigned a. value from D2 to DO. For case statement we CB Ai 1 1 0 0 1 0 0 0 1 I (I) l 8(0) 10 1 1 0 I 0 1 0 (0) 0 0, 0l (B') 1 I 0 1 I 0 0 1 1 (1) 0 (0) 0. 1 Ao B1 Bo B'0 A1AoB1 1----1-------1----1-------- 1----+t+------ 1011 ll 00 1-----ttt..-------;-r1-1101 --------t--H-- B' 00 1110 1---+H-H.----H'+-ll 11 1----H+I-H----H-~ Bo ?(Bo) 00101--------'-l--OOll 1------'----'-l--OIOO t----e----..,.-+--0101 0110 0111 - - - - - - - 10001---- i...~---H.;...._1001 1 - - - - r i . - - - - H - 1010 J - - - + H - - - - - H + - - 0 0 A=B B' 10 Bo A1A0 B1. A>B Solution using 4 to 16 decoder Solution using 8 to 1 multiplexers Though we are not presenting them as a separate method, the AND bank (inside decoder) and OR bank combination concept presented here can be used to obtain solution from programmable logic devices such as PLA, PAL, etc. A=B 15 13 12 JO 1 14 11 9 IC 7485 4 0 3 2 0 In Method:..4, we follow a straightfor,_.;...__~5_ _~6;;..________-'---' ward approach to use a 4-bit comparator{IC 7485 : Fig. 4.38a}for the purpose as shown A=B in Fig. 4.53. We keep the higher two bits 'O' Solution using 4-bit comparator and 'A = B input' high so that it essentially becomes a 2-'bit comparator generating all three outputs A > B, A = B and A < B of which only first .two are useful here. Data-Processing Circuits Amultiplexer is a circuit with many inputs but only one output. The 16-to-1 multiplexer has 16 input bits, 4 control bits, and 1 output bit. The 4 control bits select and steer 1 of the 16 inputs to the output. The multiplexer is a universal logic circuit because it can generate any truth table. A demultiplexer has one input and many outputs. By applying control signals, we can steer the input signal fo ·one of the output lines. A decoder is similar to a demultiplexer, except that there is no data input. The control bits are the only input. They are decoded by activating one of the output lines. BCD is an abbreviation for binary-coded decimal. The BCD code expresses each digit in a decimal number by its nibble equivalent. A BCD-to-decimal decoder converts a BCD input to its equivalentdecimal value. A seven-segment decoder converts a BCD input to an output suitable for driving a seven-segment indicator. An encoder converts an input signal into a coded output signal. An example is the decimal-to-BCD encoder. An exclusive-OR gate has a high output only when a~ odd number of inputs are high. Exclusive:OR gates are useful in parity generators-checkers. Magrutude comparators are useful in comparing two binary numbers. It generates three outputs that give if one number is greater, equal or less than the other number. Cascading magnitude comparators we can compare two numbers of any size. A ROM is a read-only memory. Smaller ROMs are used to implement truth tables. ROMs are expensive because they require a mask for programming. PROMs are user-programmable and ideal for small production runs. EPROMs are not only user-programmable, but they are also erasable and reprogrammable during the design and development cycle. PALs are chips that.are programmable arrays of logic. Unlike the PROM with its fixed AND array and programmable OR array, a PAL has programmable AND array and a fixed OR array. The PAL has the advantage ofhaving up to 16 inputs in commercially available devices. In the PLA both the AND array and the OR array are programmable. The PLA is a much more versatile programmable logic device (PLD) IC than the PROM or the PAL. • active low The low state is the one that causes something to happen rather than the high state. • BCD A binary-coded decimal. • data selector A synonym for multiplexer. • decoder A circuit that is similar to a demultiplexer, except there is no data input. The control input bits produce one active output line. • demultiplexer A circuit with one input and many outputs. • EPROM An erasable programmable readonly memory. With this device, the user can erase the stored contents with ultraviolet light and electrically store new data. EPROMs are useful during project development where programs and data are being perfected. " even parity A binary number with an even number of 1s. • exclusive-OR gate A gate that produces a high output only when an odd number of inputs is high. • LED A light-emitting diode. • logic probe A troubleshooting device that indicates the state of a signal line. " Magnitude comparator compares two binary numbers and signals if one is greater, equal or less than other. • multiplexer A circuit with many inputs but only one output. • odd parity A binary number with an odd number of 1s. • PAL A programmable array logic (sometimes written PLA, which stands for programmable Digital Principles and Applications logic array). In either case, it is a chip with a programmable AND array and a fixed OR aITay. 11 parity generation An extra bit that is generated and attached to a binary number, so that the new number has either even or odd parity. " PLA A programmable logic array. " PLD A programmable logic device. " PROM A programmable read-only memory. A type of chip that allows the user to program it with a PROM programmer that bums fusible 4.1 In Fig. 4.2, if ABCD = 1001, what does Y equal? 4.2 In Fig. 4.4, if ABCD = llOO, what does Y equal? 4.3 We want to implement Table 3.12 of the preceding chapter using multiplexer logic. Show a circuit, similar to the one in Fig. 4.4, that can do the job. A B C 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 I 0 I 1 1 J 1 1 1 1 I 0 0 0 0 1 I I 0 0 I 0 1 0 I 0 I 0 0 I 0 0 0 I 0 I 1 0 0 1 0 0 I 1 1 0 I 0 0 0 1 1 0 0 0 0 0 0 0 I 0 0 0 0 1 0 0 0 0 I 0 0 0 0 1 0 0 links at the diode cross points. Once the data is stored, the programming is permanent. PROMs are useful for small production runs. " ROM A read-only memory. An IC that. can store many binary numbers at locations called addresses. ROMs are expensive to manufacture and are used only for large production runs where the cost of the mask can be recovered by sales. • strobe An input that disables or enables a circuit. 4.4 Show how to connect a 74150 to implement this Boolean equation: y Y1 Yo A J 0 0 0 0 0 0 0 0 I I 0 0 0 1 0 0 0 0 0 0 0 0 I 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 = ABCD + ABCD + ABCD 4.5 Draw a circuit with four 74150s that has a truth table like the one in Table 4.11. 4.6 Table 4.12 shows the Gray code. Show how four 74150s may be connected to convert from binary to Gray code. Show how the same can be realized by four 74151 ICs (8-tol multiplexer). Yo 0 0 0 0 ,Q 0 0 l J l 1 1 l I 1 l 0 0 0 0 l I 0 0 l 1 0 0 1 l 0 0 1 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 I 0 I 0 1 0 1 I I 1 l I 0 1 1 0 0 I 0 0 1 I 1 1 1 0 0 l 1 l 1 0 0 I 1 0 0 0 0 0 l 1 0 0 Data-Processing Circuits 4.7 In Fig. 4.12, if ABCD = 0101, which is the active output line when the strobe is high? When it is low? 4.8 Input signals R and T are low in Fig. 4.13. Which is the active output line when ABCD = 0011? To have the Y9 output line active, what input signals do you need? 4.9 Suppose a logic probe shows that pin 19, given in Fig. 4.13, is always high. Which of the following may possibly cause trouble: a. Pin 20 is grounded. b.Pin 18 has a sine wave instead of pulses. c. The R input is grounded. d. The T input is connected to +5 V. 4.10 Are the output signals of Fig. 4.15 active low or active high? For the IC to decode the ABCD input, does the strobe have to be low or high? 4.11 In Fig. 4.16, supposeX= 1 andABCD = 0110. Which is the active chip and which is the active output line? 4.12 Design a circuit that realizes following two functions using a decoder and two OR gates. ChipO 16 Vee Yo Y1 Yz 12 A 13 B 14 C 15 Y3 Y4 Ys y6 D 8 - Y7 Yg GND Y9 F1(A,B,C) = L m(l,3,7), F2(A,B,C) = L m(2,3,5) and F3(A,B,C) = L m(O, 1,5,7) 4.14 Convert the following decimal numbers into their BCD equivalents: a. 32 b. 634 C. 4898 4.15 Convert the following BCD numbers into their decimal equivalents: a. 0110 0111 b. 1000 0001 0011 C. 0111 0010 0101 1001 4.16 In Fig. 4.18, what is the high output line when ABCD=Ol01?

4.17 In Fig. 4.20, which is the low output when ABCD = 0111? 4.18 Figure 4.54 shows a group of chips numbered 0 through 9. Each chip has an active-low STROBE input. Which chip is active for each of these conditions: Chip 1 STROBE +5V F 1(A,B) = L m(0,3) and F2(A,B) =Lm(l,2) 4.13 Design a circuit that realizes following three functions using a decoder and three OR gates. Chip2 STROBE ·········· I 111 II 111111 I 11111 111111 111111 111 I I 1111 111 11 I Chip9 STROBE Digital Principles and Applications a. ABCD = 0000. b. ABCD = 0010. c. ABCD = 1001. 4.19 The ABCD input of Fig. 4.54 initially equals 1111. For this condition, all output waveforms start high in the timing diagram of Fig. 4.55. Another circuit not shown is supposed to produce the following input values of ABCD: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, and 1001. Yo Y, --U LJ Y2 Y3 Y4 Ys y6 Y7 Yg Y9 The timing diagram tells us that something is wrong with the logic circuit of Fig. 4.54. Which of the following is a possible trouble: a. Pin 16 is not connected to the supply b. Pin 8 is open. voltage. c. Pin 12 is short-circuited to the ground. d.Pin 15 is short-circuited to +5 V. 4.22 In Fig. 4.25, what is the output when button 7 is pressed? When button 3 is pressed? 4.23 In Fig. 4.27, if button 8 is pressed, which is the input pin that goes into the low state? What does the ABCD output equal? 4.24 In Fig. 4.32d, what does Y equal for each of the following inputs: a. 000110 b. 011001 C. 011111 d. 111100 4.25 In Fig. 4.33, what does Y equal for the following inputs: a. 1111 0000 0000 1111 b. 0101 1010 0111 1100 0001 C. 1110 1011 1101 d. 0001 0101 0110 0011 4.26 In Fig. 4.56, the 8-bit register is a logic circuit that stores byte A 1 ... A 0 . What does byte Y1 ... Yo equal for each of these conditions: a. A1 ... Ao= 1000 0111 and INVERT= 0. b. A1 ... Ao= 0011 1100 and INVERT= I. c. A 7 ... Ao= 1111 0000 and INVERT= 0. d. A 7 ... Ao= 1110 0001 and INVERT= 1. 8-bit register 4.20 In Fig. 4.21, which of the segments have to be active to display the following digits: a. 2 C. 8 b. 6 4.21 In Fig. 4.23a, Vcc = +5 V, all resistors are 1 k.Q, and each LED has a voltage drop of 2 V. Approximately how much cun-ent is there through an active segment? 4.27 In Fig. 4.57 on the next page, each register is a logic circuit that stores a 6-bit number. The left register stores A 5 ••• Ao and the right register stores B 5 ..• B 0 . What value does the output signal labeled EQUAL have for each of these: Data-Processing Circuits A register B register A5 A4 A 3 A 2 A 1 A0 B5 B4 B3 B2 B 1 B0 TFFr~~~o EQUAL a. A 7 ••• Ao is less than B1 ... Bo. b. A1 ... Ao equals B1 ... Bo. c. A1 ... Ao is greater than B1 ... B+ 4.28 In Fig. 4.58, what does X8 equal for each of the following X 7 ••• Xo inputs: a. 0000 1111 b. 1111 0001 C. 1010 1110 d. 1011 1100 4.29 In Fig. 4.58, what changes can you make to get a 9-bit output with even parity? +5 V 14 4.30 In Fig. 4.58, assuming the circuit is working all right, what will the logic probe indicate for each of the following: a. Input data has even parity. b. Input data has odd parity. c. Pins 3 and 4 are grounded. 4.31 Write the (X > Y) equation for a 4-bit comparator. 4.32 Show how magnitude of two IO-bit numbers can be compared using IC 7485. 4.33 Suppose a ROM has 8 input address lines. How many memory locations does it have? 4.34 Two 74S370s are connected in parallel. To address all memory locations, how many bits must the binary address have? 4.35 In Fig. 4.40, ifABC= 011, what does Y3Y2Y1 Yo equal? 4.36 Draw a ROM circuit similar to the one in Fig. 4.40 that produces these outputs: 1 2 8 9 10 11 12 13 : ODDINPUT EVEN INPUT 74180 7 ~ODD OUTPUT Y3 =ABC Y2 ABC+ABC Yi= ABC+ ABC+ ABC Yo =ABC+ ABC+ ABC+ ABC Digital Principles and Applications 4.37 Draw a PROM circuit similar to the one in Fig. 4.41 that generates the Y3 to Yo output given in Table 4.11. 4.38 What is the Boolean equation for Y3 in Fig. 4.59 on the previous page? For Y2? For Yi? For Yo? 4.39 Draw a 4-input and 4-output PAL circuit that has the truth table of Table 4.11. A B C 4.40 Write. the Boolean expression for the output Y3 in Fig. 4.42. 4.41 The input to the PLA in Fig. 4.47 is ABCD = 0011. What segments of the indicator are illuminated and what decimal number is displayed? What if ABCD = 1001? What about 1111? 4.42 Will there be any ambiguity ifsegmentgofthe 7-segment indicator in Fig. 4.47 is defective (burned out)? What numbers are displayed? D RiRiKl ~ Programmable AND array Fixed OR array Data-Processing Circuits AIM:. The· aim of this. experiment is to display one of two BCD numbers in a 7-..segment display, Theory: .·. . . . BCffm.~bt!rs ca!lbe selected by activating select line of a multiplexer. The multiplexer output then is one of the two BCD numbers. These outputs can be connected tO four inputs of a ?-segment decoder/driver. The outputs of this driver can be connected to a ?-segment display to display.the decimal equivalent of the BCD number selected. Apparatus: 5 V DC Power supply, Multimeter, and Bread Board L 2. 3. 4. Multiplexer It means that active low. Demultiplexer ASCD= HLHL It will be high since STROBE is high. 6. The outputs are active low. Y10 is low; all other outputs are high. 8. BCD stands for binary-coded decimal. Work element: Verify the truth table of multiplexerIC 74157. Note that STROBE is an input .and find its use. The common select line applies to all four 2-to- l multiplexers. Verify the truth table ofIC7446 for BCD inputs. Select ~ BCD input resistance values to be connected in the range 220 to 1000 ohm. This is to ensure that entire power supply voltage does not drop across the LED of display. Interconnect properly all the different units and verify. 9. IO. 11. 12. LED stands for light-emitting diode. See Fig. 4.21. Each segment is an LED. An encoder converts an active input signal into a coded output signal, for instance, decimal to binary. 13. The TTL 74147 is a decimal-to-binary encoder. Digital Principles and Applications 14. The output for an exclusive-OR gate is high only when an odd number of inputs are high. 15. See Fig. 4.30. 16. There are an even number of ls (highs). 17. True. 18. Three:X= Y,X> YandX< Y. 19. Three. 20. ROM stands for read-only memory. 21. A 512 x 8 ROM is arranged as 512 eight-bit words. 22. PROM stands for programmable readonly memory. 23. PAL stands for programmable array logic. 24. The AND array is programmable; the OR array is fixed. 25. PLA stands for programmable logic array. 26. In a PLA, both the AND array and the OR array are programmable. 27. Decimal 3; segments abcdg Number Systems and Codes + + + + + Convert decimal numbers to binary and convert binary numbers to decimal Convert binary and decimal numbers to octal and convert octal numbers to binary and decimal Convert binary and decimal numbers to hexadecimal and convert hexadecimal numbers to binary and decimal Describe the ASCII code, excess-3 code, and Gray code Understand Error Detection and Correction Code 5.1 BINARY NUMBER SYSTEM The bina,y number system is a system that uses only the digits O and I as codes. All other digits (2 to 9) are thrown away. To represent decimal numbers and letters of the alphabet with the binary code, you have to use different strings of binary digits for each number or letter. The idea is similar to the Morse code, where strings of dots and dashes are used to code all numbers and letters. What follows is a discussion of decimal and binary counting. Decimal Odometer To understand how to count with binary numbers, it helps to review how an odometer (miles indicator of a car) counts with decimal numbers. When a car is new, its odometer starts with 00000 After 1 km the reading becomes 00001 Digital Principles and Applications Successive kms produce 00002, 00003, and so on, up to 00009 A familiar thing happens at the end of the tenth km. When the units wheel turns from 9 back to 0, a tab on this wheel forces the tens wheel to advance by 1. This is why the numbers change to 00010 Reset-and-Carry The units wheel has reset to Oand sent a carry to the tens wheel. Let's call this familiar action reset and cany The other wheels of an odometer also reset and carry. For instance, after 999 kms the odometer shows 00999 What does the next km do? The units wheel resets and caiTies, the tens wheel resets and carries, the hundreds wheel resets and carries, and the thousands wheel advances by 1, to get 01000 Binary Odometer Visualize a binary odometer as a device whose wheels have only two digits, 0 and 1. When each wheel turns, it displays 0, then 1, then back to 0, and the cycle repeats. A four-digit binary odometer starts with 0000 (zero) 0001 (one) After 1 mile, it indicates The next mile forces the units wheel to reset and carry, so the numbers change to 0010 (two) The third mile results in OOll 4-Digit Binary Numbers (three) Binary After 4 miles, the units wheel resets and carries, the second wheel resets and carries, and the third wheel advances by 1: 0100 0 1 2 (four) Table 5.1 shows all the binary numbers from OOOOto l ll 1, equivalent to decimal O to 15. Study this table carefully and practice counting from 0000 to 1111 until you can do it easily. Why? Because all kinds oflogic circuits are based on counting from 0000 to 1111. The word bit is the abbreviation for binary digit. Table 5.1 is a list of 4-bit number from 0000 to 1111. When a binary number has 4 bits, it is sometimes called a nibble. Table 5.1 shows 16 nibbles (0000 to llll). A binary number with 8 bits is known as a byte; this has become the basic unit of data used in computers. You will learn Decimal 3 OHO 0111 1000 1001 1010 nm HOO 1101 1110 llll 4 5 6 7 8 9 10 11 12 13 14 15 NumberSystems and Codes more about bits, nibbles, .and bytes in later chapters. For now memorise these definitions: bit =X nibble = XXXX byte = XXXXXXXX where the X may be a O oral. 1. What is the binary number~or dycimal 13? 2. Whatis the decimal.equivalent ofbim1ry 1001? 3. How many binary digits (bits)are required to represent decimal 15? 5.2 BINARY-TO-DECIMAL CONVERSION Table 5.1 lists the binary numbers from 0000 to 111 L But how do you convert larger binary numbers into their decimal values? For instance, what does binary 101001 represent in decimal numbers? This section shows how to convert a binary number quickly and easily into its decimal equivalent. Positional Notation and Weights We can express any decimal integer (a whole number) in units, tens, hundreds, thousands, and so on. For instance, decimal number 2945 may be written as 2945 = 2000 + 900 + 40 + 5 In powers of 10, this becomes 2945 = 2(10 3) + 9(10 2) + 4(10 1) + 5(10°) The decimal number system is an example of positional notation, each digit position has a weight or value. With decimal numbers, the weights are units, tens, hundreds, thousands, and so on. The sum of all the digits multiplied by their weights gives the total amount being represented. In the foregoing example, the 2 is multiplied by a weight of 1000, the 9 by a weight of 100, the 4 by a weight of 10, and the 5 by a weight of 1; the total is 2000 + 900 + 40 + 5 = 2945 Binary Weights In a similar way, we can rewrite any binary number in terms of weights. For instance, binary number 111 becomes 111=100+10+1 (5.1) In decimal numbers, this may be rewritten as 7=4+2+1 (5.2) Writing a binary number as shown in Eq. (5 .1) is the same as splitting its decimal equivalent into units, 2s, and 4s as indicated by Eq. (5.2). In other words, each digit position in a binary number has a weight. The least Digital Principles and Applications significant digit (the one on the right) has a weight of 1. The second position from the right has a weight of 2; the next, 4; and then 8, 16, 32, and so forth. These weights are in ascending powers of 2; therefore, we can write the foregoing equation as 2 Binary System Bit Position Weight 1 • (Right most) 1 2 4 8 16 32 1 7= 1(2)+ 1(2)+ 1(2°) Whenever you look at a binary number, you can find its decimal equivalent as follows: 3 1. When there is a 1 in a digit position, add the weight of that position. 2. When there is a Oin a digit position, disregard the weight of that position. For example, binary number 101 has a decimal equivalent of 64 128 4+0+1=5 As another example, binary number 1101 is equivalent to 8+4+0+1=13 Still another example is 1100 l, which is equivalent to 16 + 8 + 0 + 0 + 1 = 25 Streamlined Method We can streamline binary-to~decimal conversion by the following procedure: 1. 2. 3. 4. Write the binary number. Directly under the binary number write 1, 2, 4, 8, 16 ... , working from right to left. If a zero appears in a digit position, cross out the decimal weight for that position. Add the remaining weights to obtain the decimal equivalent. As an example of this approach, let us convert binary 101 to its decimal equivalent: STEP 1 STEP 2 STEP 3 STEP4 101 42 1 42 1 4 + 1= 5 As another example, notice how quickly 10101 is converted to its decimal equivalent: 1 0 16 g 1 0 4 2 1 1 -+ 21 Fractions So far, we have discussed binary integers (whole numbers). How are binary fractions converted into corresponding decimal equivalents? For instance, what is the decimal equivalent of0.101? In this case, the weights of digit positions to the right of the binary point are given by ~ , and so on. In powers of 2, 1 the weights are t, t, t, _ _ _ _ _ _ _ _ _ _ _ _ _ _ _N_u_m_b_er_S_y_st_em_s_a_n_d_C_od_e_s _ _ _ _ _ _ _ _ _ _ _ _ _ or in decimal form: 0.5 0.25 Powers of 2 0.125 0.0625 etc. Abbreviation Here is an example. Binary fraction 0.101 has a decimal equivalent of 0.1 0 1 2 4 1 0.5 + O+ 0.125 = 0625 Another example, the decimal equivalent of 0.1101 is 0.1 1 0 1 8 16 32 64 128 256 512 1,024 2,048 4,096 8,192 16,384 32,768 65,536 131,072 262,144 524,288 1,048,576 2,097,152 4,194,304 0.5 + 0.25 + 0 + 0.0625 = 0.8125 Mixed Numbers For mixed numbers (numbers with an integer and a fractional part), handle each part according to the rules just developed. The weights for a mixed number are etc. 23 22 ~ 2 1 20 · T 1 T 2 T 3 etc. i Binary point 64K 128K 256K 5I2K l,024K=lM 2,048K=2M 4,096K::4M For future reference, Table 5.3 lists powers of 2 and their decimal equivalents and the numbers ofK and M. The abbreviation K stands for 1024. Therefore, lK means 1024. 2K stands for 2048, 4K represents 4096, and so on. The abbreviation M stands for 1,048,576, which is equivalent to 1024K (1024 x 1024 = 1,048,576). A memory chip that stores 4096 bits is called a "4K memory." A digital device might have a memory capacity of 4,194,304 bytes. This would be referred to as a "4-megabyte (Mb) memory." Convert binary 110.001 to a decimal number. 0 % 0.125 .""'7 6.125 What is the decimal value of binary l O11.11? 2 A computer has a 2 Mb memory. What is the decimal equivalent of 2 Mb? Digital Principles and Applications Solution This means that the computer can store 2,097,152 bytes iri its· memory. 4. What is the decimal equivalent of 1()010? 5. What is the binary equivalentof35'? 6. A binary number has 9 bits. What is the binary weight of the most significantbit'? 5.3 DECIMAL-TO-BINARY CONVERSION One way to convert a decimal number into its binaiy equivalent is to reverse the process described in the preceding section. For instance, suppose that you want to convert decimal 9 into the corresponding binary number. All you need to do is express 9 as a sum of powers of 2, and then write 1s and Os in the appropriate digit positions: 9=8+0+0+1 -:-7 1001 As another example: 25 = 16 + 8 + 0 + 0 + 1 -:-7. 11001 Double Dabble A popular way to convert decimal numbers to binary numbers is the double~dabble method. In the doubledabble method you progressively divide the decimal number by 2, writing down the remainder after each division. The remainders, taken in reverse order, form the binary number. The best way to understand the method is to go through an example step by step. Here is how to convert decimal 13 to its binary equivalent Step 1 Divide 13 by 2, writing your work like this: 6 2)13 I -:-7 (first remainder) The quotient is 6 with a remainder of 1. Step 2 Divide 6 by 2 to get 3 2)6 2)13 This division gives 3 with a remainder of 0. 0 -:-7 (second remainder) l -:-7 (first remainder) Number Systems and Codes Step 3 Again you divide by 2: I 2 f3 0 ~ (second remainder) 2)6 l~ (first remainder) 2)13 1 ~ (first remainder) r Here you get a quotient of I with a remainder of I. Step 4 One more division gives 0 2)1 2)3 2)6 2)13 I 1 0 (fourth ,emainde,) Read down I In this final division 2 does not divide into 1; thus, the quotient is Owith a remainder of 1. Whenever you arrive at a quotient of O with a remainder of 1, the conversion is finished. The remainders when read downward give the binary equivalent. In this example, binary 1101 is equivalent to decimal 13. There is no need to keep writing down 2 before each division because you are always dividing by 2. Here is an efficient way to show the conversion of decimal 13 to its binary equivalent: 0 1 3 6 2)13 ~1 Read down Fractions As far as fractions are concerned, you multiply by 2 and record a carry in the integer position. The carries read downward are the binary fraction. As an example, 0.85 converts to binary as follows: 0.85 x 2 = 1.7 = 0.7 with a carry of l 0.7 x 2 = 1.4 = 0.4 with a carry of 1 0.4 x 2 = 0.8 = 0.8 with a carry ofO 0.8 x 2 = 1.6 = 0.6 with a carry of 1 0.6 x 2 = 1.2 = 0.2 with a carry of 1 0.2 x 2 = 0.4 = 0.4 with a carry of 0 Read down Reading the carries downward gives binary fraction 0.110110. In this case, we stopped the conversion process after getting six binary digits. Because of this, the answer is an approximation. If more accuracy is needed, continue multiplying by 2 until you have as many digits as necessary for your application. Digital Principles and Applications Useful Equivalents Table 5.4 shows some decimal-binary equivalences. This will be useful in the future. The table has an important property that you should be aware of. Whenever a binary number has all 1s (consists of only 1s), you can find its decimal equivalent with this formula: Decimal = 211 - 1 where n is the number of bits. For instance, 1111 has 4 bits; therefore, its decimal equivalent is Decimal = 24 - 1 = 16 -'- I = 15 Decimal-Binary Equivalences Decimal Binary l 3 11 7 15 31 63 127 255 511 1,023 2,047 4,095 8,191 16,383 32,767 65,535 111 1111 I 1111 11 1111 111 lll I lll! 1111 1 llll 1111 111111 llll 111 llll lll! ll 11 ll 1111 l llll llll 1111 ll llll 11111111 lll 1111 1111 llll 1111 1111 llll 1111 As another example, 1111 1111 has 8 bits, so Decimal= 28 - 1 =256-1 = 255 BCD-8421 and BCD-2421 Code Binary Coded Decimal (BCD) refers to representation ofdigits 0-9 in decimal system by 4-bit unsigned binary numbers. The usual method is to follow 8421 encoding which employs conventional route of weight placements like 8 representing the weight of the 4th. place (as 24- 1 = 8), 4, i.e. 23- 1 of the 3rd place, 2, i.e. 22- 1 of the 2nd place and 1, i.e. 2 1- 1 of the 1st place. The 2421 code is similar to 8421 code except for the fact that the weight assigned to 4th place is 2 and not 8. The decimal numbers 0-9 in these two codes then can be represented as shown in Table 5.5. BCD-8421 and BCD-2421 Code Decimal BCD-8421 BCD-2421 0 0000 0001 0010 0011 0100 0101 0110 Olll 1000 1001 0000 0001 0010 1 2 3 4 5 6 7 8 9 0011' 0100 1011 HOO 1101 1110 llll Number Systems and Codes As an example, decimal number 29 in BCD-8421 is written as 00101001 (0010 representing 2 and 1001 representing 9) while in BCD-2421, it is written as 00101111 (0010 representing 2 and 1111 representing 9). Convert decimal 23.6 to a binary number. Split decimal 23.6 into an integer of 23 ·and a fraction of 0.6, arid apply double dabble to each 0 1 1 0 2 5 11 2)23 Read down 1 l and 0,6 x 2 = 1.2 = 0.2 with a carry ofl 0.2 x 2 = 0.4 = 0.4with a carry ofO 0.4 x 2 = 0.8 = 0.8 with a carry ofO Read down 0.8 x 2 = L6 = 0.6 with a carry ofl 0.6 x 2 = 0.2 0.2 vvith a carry of 1 The binary number is 10111.1001 L This 10-bit numberis an approximation of decimal 21.6 because we terminated the conversion of the fractional part after 5 bits. A digital computer processes binary numbers that are 32 bits long. If a 32-bit number has all 1s, what is its decimal equivalent? Solution Decimal a:231 - =(28)(28)(28)(28)-· I =(256)(256)(256)(256)-l 4,294,967,295 7. Whatis double dabble? A binafy nmnber is composed of twelve Ts. What is its decimal equivalent? 9. What is the binary number for decimal 255? 5.4 OCTAL NUMBERS The base of a number system equals the number of digits it uses. The decimal number system has a base of 10 because it uses the digits O to 9. The binary number system has a base of 2 because it uses only the digits 0 and I. The octal number system has a base of 8. Although we can use any eight digits, it is customary to use the first eight decimal digits: 0, 1,2,3,4,5,6, 7 Digital Principles and Applications (There is no 8 or 9 in the octal number code.) These digits, 0 through 7, have exactly the same physical meaning as decimal symbols; that is, 2 stands for••, 5 symbolizes ..... , and so on. Octal Odometer The easiest way to learn how to count in octal numbers is to use an octal odometer. This hypothetical device is similar to the odometer of a car, except that each display wheel contains only eight digits, numbered O to 7. When a wheel turns from 7 back to 0, it sends a carry to the next-higher wheel. Initially, an octal odometer shows 0000 (zero) 0001 0002 0003 0004 0005 0006 0007 (one) (two) (three) (four) (five) (six) (seven) The next 7 kms produces readings of At this point, the least-significant wheel has run out of digits. Therefore, the next km forces a reset and carry to obtain 0010 (eight) The next 7 kms produces these readings: 0011, 0012, 0013, 0014, 0015, 0016, and 0017. Once again, the least-significant wheel has run out of digits. So the next km results in a reset and carry: 0020 (sixteen) Subsequent kms produce readings of 0021, 0022, 0023, 0024, 0025, 0026, 0027, 0030, 0031, and so on. You should have the idea by now. Each km advances the least-significant wheel by one. When this wheel runs out of octal digits, it resets and carries. And so on for the other wheels. For instance, if the odometer reading is 6377, the next octal number is 6400. Octal-to-Decimal Conversion How do we convert octal numbers to decimal numbers? In the octal number system each digit position corresponds to a power of 8 as follows: g3 g2 gI gO. g-1 g-2 g-3 i Octal point Therefore, to convert from octal to decimal, multiply each octal digit by its weight and add the resulting products. Note that s0 = I. For instance, octal 23 converts to decimal like this: 2(8 1) + 3(8°) = 16 + 3 = 19 Number Systems and Codes Here is another example. Octal 257 converts to 2(8 1)+5(8 1)+7(8°)= 128+40+7= 175 Decimal-to-Octal Conversion How do you convert in the opposite direction, that is, from decimal to octal? Octal dabble, a method similar to double dabble, is used with octal numbers. Instead of dividing by 2 (the base of binary numbers), you divide by 8 (the base of octal numbers) writing down the remainders after each division. The remainders in reverse order form the octal number. As an example, convert decimal 175 as follows: 0 8)2 2 ~ (third remainder) 8 }"TI 8)175 5 ~ (second remainder) 7 ~ (first remainder) You can condense these steps by writing 0 2 21 8)175 2~ ·1 Read down Thus decimal 175 is equal to octal 257. Fractions With decimal fractions, multiply instead of divide, writing the carry into the integer position. An example of this is to convert decimal 0.23 into an octal fraction. 0.23 x 8 = 1.84 = 0.84 with a carry of 11 0.84 x 8 = 6.72 = 0.72 with a carry of6 Read down 0.72 x 8 = 5.76 = 0.76 with a carry of5 etc. The carries read downward give the octal fraction 0.165. We terminated after three places; for more accuracy, we would continue multiplying to obtain more octal digits. Octal-to-Binary Conversion Because 8 (the base of octal numbers) is the third power of 2 (the base of binary numbers), you can convert from octal to binary as follows: change each octal digit to its binary equivalent. For instance, change octal 23 to its binary equivalent as follows: 2 3 t t 010 Oll Here, each octal digit converts to its binary equivalent (2 becomes O10, and 3 becomes O11). The binary equivalent of octal 23 is O10 011, or O10011. Often, a space is left between groups of 3 bits; this makes it easier to read the binary number. Digital Principles and Applications As another example, octal 3574 converts to binary as follows: 3 5 7 4 J, J, J, J, 011 101 111 100 Hence binary O11101111100 is equivalent to octal 3574. Notice how much easier the binary number is to read ifwe leave a space between groups of3 bits: 011 101 111 100. Mixed octal numbers are no problem. Convert each octal digit to its equivalent binary value. Octal 34.562 becomes 3 4 5 6 2 J, J, J, J, J, 011 100 101 110 010 Binary-to-Octal Conversion Conversion from binary to octal is a reversal of the foregoing procedures. Simply remember to group the bits in threes, starting at the binary point; then convert each group of three to its octal equivalent (Os are added at each end, if necessary). For instance, binary number 1011.01101 converts as follows: 1011.01101 ~ 001 011. 011 J, J, 3 1 010 J, 3 J, 2 Start at the binary point and, working both ways, separate the bits into groups of three. When necessary, as in this case, add Os to complete the outside groups. Then convert each group of three into its binary equivalent. Therefore: 1011.01101 = 13.32 The simplicity of converting octal to binary and vice versa has many advantages in digital work. For one thing, getting information into and out of a digital system requires less circuitry because it is easier to read and print out octal numbers than binary numbers. Another advantage is that large decimal numbers are more easily converted to binary if first converted to octal and then to binary, as shown in Example 5.6. What is the binary equivalent of decimal 363? Solution One approach is double dabble. Another approach is octal dabble, followed by octal-to~binary conversion. Here is how the second method works: :J Next, convert octal 553 to its binary equivalent: 5 J, 101 3 J, 101 The double-dabble approach would produce the same answer, hut it is tedious because you h,aye to divide by 2 times before the conversion terminates. Number Systems and Codes 10. What are the digits used in the octal number system? What is the oc~l,numberfor .binary. I ll? What is the decimal number. for binary 111? 5.5 HEXADECIMAL NUMBERS Hexadecimal numbers are used extensively in microprocessor work. To begin with, they are much shorter than binary numbers. This makes them easy to write and remember. Furthermore, you can mentally convert them to binary whenever necessary. The hexadecimal number system has a base of 16. Although any 16 digits may be used, everyone uses 0 to 9 and A to F as shown in Table 5.6. In other words, after reaching 9 in the hexadecimal system, you continue counting as follows: Decimal Binwy Hexadecimal 0 l 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111 0 2 3 4 5 6 7 A,B,C,D,E,F 8 9 10 Hexadecimal Odometer The easiest way to learn how to count in hexadecimal numbers is to use a hexadecimal odometer. This hypothetical device is similar to the odometer of a car, except that each display wheel has 16 digits, numbered 0 to F. When a wheel turns from F back to 0, it sends a carry to the next higher wheel. Initially, a hexadecimal odometer shows 0000 The next 9 kms produces readings of Hexadecimal Digits 11 12 13 14 15 l 2 3 4 5 6 7 8 9 A B C D E F (zero) The next 6 kms gives OOOA (ten) (two) OOOB (eleven) (three) oooc (twelve) 0004 (four) OOOD (thirteen) 0005 (five) OOOE (fourteen) 0006 (six) OOOF (fifteen) 0007 (seven) 0008 (eight) 0009 (nine) 0001 (one) 0002 0003 Digital Principles and Applications At this point, the least-significant wheel has run out of digits. Therefore, the next km forces a reset and carry to obtain 0010 (sixteen) The next 15 kms produces these readings: 0011, 0012, 0013, 0014, 0015, 0016, 0017, 0018, 0019, 001A, OOIB, OOlC, 001D, OOlE, and OOlF. Once again, the least significant wheel has run out of digits. So, the next km results in a reset and carry: 0020 (thirty-two) Subsequentkmsproducereadingsof0021,0022,0023,0024,0025,0026,0027,0028,0029,002A,002B, 002C, 002D, 002E, and 002F. You should have the idea by now. Each km advances the least-significant wheel by one. When this wheel runs out of hexadecimal digits, it resets and carries, and so on for the other wheels. For instance, here are three more examples: Next number 835D A480 Number 835C A47F BFFF cooo Hexadecimal-to-Binary Conversion To convert a hexadecimal number to a binary number, convert each hexadecimal digit to its 4-bit equivalent using the code given in Table 5.5. For instance, here's how 9AF converts to binary: 9 A F J, J, J, 1001 1010 1111 As another example, C5E2 converts like this: J, E J, J, 0101 1110 0010 C 5 J, 1100 2 Binary-to-Hexadecimal Conversion To convert in the opposite direction, from binary to hexadecimal, again use the code from Table 5.5. Here are two examples. Binary 1000 llOO converts as follows: 1000 llOO J, J, 8 C Binary 1110 1000 1101 0110 converts like this: 1110 1000 1101 0110 J, J, 8 J, J, D 6 E Number Systems and Codes In both these conversions, we start with a binary number and wind up with the equivalent hexadecimal number. Hexadecimal-to-Decimal Conversion How do we convert hexadecimal numbers to decimal numbers? In the hexadecimal number system each digit position corresponds to a power of 16. The weights of the digit positions in a hexadecimal number are as follows 16 3 16 2 16 1 16° . 16- 1 16-2 16-3 i Hexadecimal point Therefore, to convert from hexadecimal to decimal, multiply each hexadecimal digit by its weight and add the resulting products. Note that 16° = 1. Here's an example. Hexadecimal F8E6.39 converts to decimal as follows: F8E6 = F(16 3) + 8(162) + E(16 1) + 6(16°) + 3(16- 1) + 9(16-2) = 15(163) + 8(162) + 14(16 1) + 6(16°) + 3(16- 1) + 9(16-2) = 61,440 + 2048 + 224 + 6 + 0.1875 + 0.0352 = 63,718.2227 Decimal-to-Hexadecimal Conversion One way to convert from decimal to hexadecimal is the hex dabble. The idea is to divide successively by 16, writing down the remainders. Here's a sample of how it's done. To convert decimal 2479 to hexadecimal, the first division is 154 15 ~F 16)2479 In this first division, we get a quotient of 154 with a remainder of 15 (equivalent to F). The next step is lO~A 15 ~F 9 154 16)2479 l Here we obtain a quotient of9 with a remainder of 10 (same as A). The final step is 0 9 154 9~9 lO~A 15~F Read down 16)2479 Therefore, hexadecimal 9AF is equivalent to decimal 2479. Notice how similar hex dabble is to double dabble. Notice also that remainders greater than 9 have to be changed to hexadecimal digits (10 becomes A, 15 becomes F, etc.). Digital Principles and Applications Using Appendix 1 * A typical microcomputer can store up to 65,535 bytes. The decimal addresses of these bytes are from O to 65,535. The equivalent binary addresses are from 0000 0000 0000 0000 1111 1111 1111 1111 The first 8 bits are called the upper byte, and the second 8 bits are the lower byte. If you have to do many conversions between binary, hexadecimal, and decimal, learn to use Appendix 1. It has four headings: binwy, hexadecimal, upper byte, and lower byte. For any decimal number between O and 255, you would use the binary, hexadecimal, and lower byte columns. Here is the recommended way to use Appendix 1. Suppose you want to convert binary 0001 1000 to its decimal equivalent. First, mentally convert to hexadecimal: ~ 0001 1000 18 (mental conversion) Next, look up hexadecimal 18 in Appendix 1 and read the corresponding decimal value from the lowerbyte column: 18 ~ 24 (look up in Appendix 1) For another example, binary 1111 0000 converts like this: 1111 0000 ~ FO ~ 240 The reason for mentally converting from binary to hexadecimal is that you can more easily locate a hexadecimal number in Appendix 1 than a binary number. Once you have the hexadecimal equivalent, you can read the lower-byte column to find the decimal equivalent. When the decimal number is greater than 255, you have to use both the upper byte and the lower byte in Appendix 1. For instance, suppose you want to convert this binary number to its decimal equivalent: 1110 1001 0 ll l 0100 First, convert the upper byte to its decimal equivalent as follows: 1110 1001 ~ E9 ~ 59,648 (upper byte) Second, convert the lower byte to its decimal equivalent: 0111 0100 ~ 74 ~ 116 (lower byte) Finally, add the upper and lower bytes to obtain the total decimal value: 59,648 + 116 = 59,764 Therefore, binary 1110 1001 0111 0100 is equivalent to decimal 59,764. Once you get used to working with Appendix 1, you will find it to be a quick and easy way to convert between the number systems. Because it covers the decimal numbers from Oto 65,535, Appendix 1 is extremely useful for microprocessors where the typical memory addresses are over the same decimal range. * A number of hand calculators will convert binary, octal, decimal and hexadecimal numbers. Number Systems and Codes A computer memory can store thousands of binary instructions and data. A typical microprocessor has 65,536 addresses, each storing a byte. Suppose that the first 16 addresses contain these bytes: OOll 1100 0101 0010 1111 0010 1101 0100 Olll llOO 1000 0010 0010 OOll OOll 0001 llOO llOl Olll 1000 0001 1010 0100 0000 0111 0011 0100 1000 0001 1010 lllO 1111 Convert these bytes to their hexadecimal equivalents. Solution Here are the stoted bytes and their hexadecimal equivalents: 00101000 llUOOOl 00101010 HOfOlOO 01000000 Digital Principles and Applications that it contains 3C. Either way, we obtain the .same information. But notice how much easier it think 3C than it is to say, write, and thinkOO H 1100. In otherwords, hexadecimal numbers are tm1ch,easiiehFornec1nle to work with. Convert the hexadecimal numbers of the preceding example to their decimal equivalents. Solution The first address contains 3C, which converts like this: 3(16 1)+C{l6°)=48+ 12=60 Even easier, look up the decimal equivalent of 3C in Appendix 1, and you get 60. Either by powers of 16 or with reforence to Appendix 1, we can convert the other memory contents to get the following: Memory contents 0011 llOO 1100 1101 01010111 00101000 ll lJ 0001 00101010 1101 0100 01000000 om om 1100 0011 10000100 00101000 00100001 0011 1010 oonn10 0001 llll Hexadecimal equivalents 3C CD 57 28 Fl 2A D4 40 77 C3 84 28 21 3A 3E Decimal equivalents 60 205 87 40 241 42 212 64 1F Convert decimal 65,535 to its hexadecimal and binary equivalents. Solution Use hex dabble as follows: 0 15 255 4095 16)65,535 15~F 1.5 . . ~.F ··1·· 15~F .15~F Read down is Therefore, decimal 65,535 equivalent to hexadecimal FF'FE Next, convert from hexadecimal to binary as follows: F llll F F i i llll 1111 This means that hexadecimal FFFF is equivalent to binary 1111 1111 · 1111 1111. Number Systems and Codes Show how to use Appendix l to convert decimal 56,000 to its hexadecimal and binary equivalents. Solution Th([! first thing to dois tolocate the largest decimal number equal to 56.000 or less in Appendix 1. The number is 55,808, which converts like this: 55,80K-,i. DA (upper byte) Next, you need to subtract this upper byte from the originalnumber: 56,000 55,808 = 192 (difference) This difference is always less than 256 and represents the lower byte, which Appendix 1 converts as follows: 192 -,i. co Now, combine the upper and lower byte to obtain DACO which you can mentally convert to binary: DACO -,i. 1101 1010 llOO 0000 Convert Table 5.4 into a new table with three column headings: "Decimal," "Binary," and "Hexadecimal." Solution This is easy. Convert each group of bits to its hexadecimal equivalent as shown in Table 5.7. Decimal-Binary-Hexadecimal Equivalences Binary, Decimal 1 3 7 15 31 63 127 255 511 1,023 2,047 4,095 8,191 16,383 32,767 65,535 1 11 111 1 lll 11 lll 111111 1111111 11111111 111111 lll lll 1111 ll l lllllllll 111111111111 lll 11111 lll ll ll ll 1111111111 111111111111 l ll 111111111 ll lll ll What are the symbols used in hexadecimalnumbers? Whatis the binary equivalent of hexadecimal 3C? What is the decimal equivalent ofhexadecimal 3C? hexadecimal l 3 7 F IF 3F 7F FF lFF 3FF 7FF FFF lFFF 3FFF 7FFF FFFF Digital Principles and Applications To get information into and out of a computer, we need to use some kind of alphanumeric code (one for letters, numbers, and other symbols). At one time, manufacturers used their own alphanumeric codes, which led to all kinds of confusion. Eventually, industry settled on an input-output code known as the American Standard Code for Information Interchange (ASCII, pronounced ask' -ee). This code allows manufacturers to standardize computer hardware such as keyboards, printers, and video displays. Using the Code The ASCII code is a 7-bit code whose format is X6X5X4X3X2X1Xo where each Xis a O or a 1. Use Table 5 .8 to find the ASCII code for the uppercase and lowercase letters of the alphabet and some of the most commonly used symbols. For example, the table shows that the capital letter A has an X05X 4 of 100 and anX3X2X1Xo ofOOOl. The ASCII code for A is, therefore, 1000001 For easier reading, we can leave a space as follows: 100 0001 (A) 110 0001 (a) 110 0010 (b) The letter a is coded as More examples are ASCII Code X6X5X4 010 SP 110 ill Q a R b p q r JOO 101 @ p A B # C C $ D E F d % & ( ) * + G H e f g h I J K L M N 0 j k l m n 0 Number Systems and Codes 1100011 110 0100 (c) (d) and so on. Also, study the punctuation and mathematical symbols. Some examples are 010 0100 010 1011 011 1101 ($) (+) (=) In Table 5.7, SP stands for space (blank). Hitting the space bar of an ASCII keyboard sends this into a microcomputer: 010 0000 (space) Parity Bit The ASCII code is used for sending digital data over telephone lines. As mentioned in the preceding chapter, 1-bit errors may occur in transmitted data. To catch these errors, a parity bit is usually transmitted along with the original bits. Then a parity checker at the receiving end can test for even or odd parity, whichever parity has been prearranged between the sender and the receiver. Since ASCII code uses 7 bits, the addition of a parity bit to the transmitted data produces an 8-bit number in this format: X1X6X5X4 X3X2X1Xo i Parity bit This is an ideal length because most digital equipment is set up to handle bytes of data. EBCDIC as Alphanumeric Code There exists few others but relatively less used alphanumeric codes. The EBCDIC is an abbreviation of Extended Binary Coded Decimal Interchange Code. It is an eight-bit code and primarily used in IBM make devices. Here, the binary codes ofletters and numerals come as an extension of BCD code. The bit assignments of EBCDIC are different from the ASCII but the character symbols are the same. With an ASCII keyboard, each keystroke produces the ASCII equivalent of the designated character. Suppose that you type PRINT X. What is the output of an ASCII keyboard? A computer sends a message to another computer using an odd-parity bit. Here is the message in ASCII code, plus the parity bit: 1100 1000 0100 0101 0100 1100 0100 1100 0100 1111 What do these numbers mean? Digital Principles and Applications Solution First, notice that each 8-bit number has odd parity, an indication that no I-bit.errors occurred during transmission. Next, use Table 5.7 to translate the ASCII.characters. If you do this correctly, you get a message of HELLO. 15. \\'hat is the ASCII code? 16. What symbol is represented by the ASCII code 100 0000? 17. \\'hat ASCIIcode is used the percent sign, %? for 5.7 THE EXCESS-3 CODE The excess-3 code is an important 4-bit code sometimes used with binary-coded decimal (BCD) numbers. To convert any decimal number into its excess-3 fonn, add 3 to each decimal digit, and then convert the sum to a BCD number. Take another example; convert 29 to an excess-3 For example, here is how to convert 12 to an number: excess-3 number. First, add 3 to each decimal digit: +3 4 1 +3 5 Second, convert the sum to BCD fom1: 4 j, 5 j, 0100 0101 2 +3 5 9 +3 12 j, j, 0101 llOO After adding 9 and 3, do not carry the 1 into the next column; instead, leave the result intact as 12, and then convert as shown. Therefore, 0101 1100 in the excess-3 code stands for decimal 29. So, 0100 0101 in the excess-3 code stands for decimal 12. Table 5.9 shows the excess-3 code: In each case, the excess-3 code number is 3 greater than the BCD equivalent. Such coding helps in BCD arithmetic as 9's complement of any excess-3 coded number can be obtained simply by complementing each bit. Take for example decimal number 2. Its 9's complement is 9- 2 = 7. Excess-3 code of2 is 0101. Complementing each bit we get 1010 and its decimal equivalent is 7. To convert BCD to excess-3 we need an adder and for the reverse we need a subtractor. These circuits are discussed in the next chapter. Incidentally, if you need an integrated circuit (IC) that converts from excess 3 to decimal, look at the data sheet of a 7443. This transistor-transistor logic (TTL) chip has four input lines for the excess-3 input and IO output lines for the decoded decimal output. 0000 0001 0010 OOll 0100 0101 0110 0111 1000 1001 0011 0100 0101 OHQ 0111 1000 1001 1010 1011 1100 Number Systems and Codes 5.S The advantage of such coding will be understood from this example. Let an object move along a track and move from one zone to another. Let the presence of the object in one zone is sensed by sensors ABC. If consecutive zones are binary coded then zone-0 is represented by ABC= 000, zone- I by ABC= 001, zone2 by ABC= 010 and so on, as shown in Fig. 5.la. Now consider, the object moves from zone-1 to zone-2. Both BC has to change to sense that movement. Suppose, sensor B (may be an electro-mechanical switch) reacts slightly late than sensor C. Then, initially ABC= 000 is sensed as if the object has moved in the other direction from zone-1 to zone-0. This problem can be more prominent if the object moves from zone-3 (ABC = 011) to zone-4 (ABC= 100) when all three sensors has to change its value. Note that, if zones are gray coded (Fig. 5.lb) such problem does not appear as between two consecutive zones only one sensor changes its value. 0 Zone No. 0 Sensor ABC 000 (Binary coded) 1 001 - 2 010 3 011 4 100 5 101 6 110 7 111 4 110 5 111 6 101 7 100 (a) oZone No. Sensor ABC (Gray coded) 0 000 1 001 I 2 011 3 010 (b) Object moving along a track with sensors: (a) Binary coded, (b) Gray coded The disadvantage with gray code is that it is not good for arithmetic operation. However, comparing truth tables of binary coded numbers and gray coded numbers (Table 5.18) we can design binary to gray converter as shown in Fig. 5.2a and gray to binary converter as shown in Fig. 5.2b. Let's see how these circuits work by taking one example each. G3 (MSB)B 3 B3 Bz G3 G2 B2 Bi GI Bi Go (LSB)B 0 (a) (b) (a) Binary to Gray converter, (b) Gray to Binary converter Consider, a binary number B3B2 B i Bo= IO ll. Following the relation shown in Fig. 5.2a we get, G3 = B3 = 1, G2 = B3 ffi B 2 = 1 ffi O= 1, G1 = B 2 ffi B 1 = 0 ffi 1 = 1 and Go = B 1 ffi Bo= 1 ffi 1 = 0, i.e. G3 G2 G1 Go = 1110 and we can verify the same from truth table. Digital Principles and Applications Similarly, for a gray coded number say, G3 G2G 1 Go = 0111 from Fig. 5.2b we get, B3 = G3 = 0, B2 = G3 EB G2 = 0 EB 1 = 1, Bi= B2 EB G1 = 1 EB l = 0 andBo = B 1 EB G0 = 0 EB 1 = 1, i.e. B 3 B2B 1B0 = 0101. Again this conversion can be verified from Table 5.10 that shows the Gray code, along with the corresponding binary numbers. Each Gray-code number differs from any adjacent number by a single bit. For instance, in going from decimal 7 to 8, the Gray-code numbers change from 0100 to 1100; these numbers differ only in the most significant bit. As another example, decimal numbers 13 and 14 are represented by Gray-code numbers 1011 and 1001; these numbers differ in only one digit position (the second position from the right). So, it is with the entire Gray code; every number differs by only 1 bit from the preceding number. Besides the excess-3 and Gray codes, there are other binary-type codes. Appendix 5 lists some of these codes for future reference. Incidentally, the BCD code is sometimes referred to as the 8421 code because the weights of the digit positions from left to right are 8, 4, 2, and other weighted codes such as the 7421, 6311, 5421, and so on. 5.9 0000 0001 OOil 0010 OllO Olll 0101 0100 1100 1101 1111 lllO 1010 1011 1001 1000 5 6 7 8 9 10 11 12 13 14 15 0110 OUJ 1000 1001 1010 1011 1100 1101 lllO nn 1. As shown in Appendix 5, there are many TROUBLESHOOTING WUHA lOGIC PULSER Figure 5.3 shows a typical logic pulser, a troubleshooting tool that generates a brief voltage pulse when its push-button switch is pressed. Because of its design, the logic pulser (on the left) senses the original state of the node and produces a voltage pulse of the opposite polarity. When this happens, the logic probe (on the right) blinks, indicating a temporary change of output state. Thevenin Circuit Push-button switch Using a logic pulser and a logic probe Figure 5.4a shows the Thevenin equivalent circuit for a typical logic pulser. The Thevenin voltage is a pulse with an amplitude of 5 V; the polarity automatically adjusts to the original state of the test node. As shown, the Thevenin resistance or output impedance is only 2Q. This Thevenin resistance is representative; the exact value depends on the particular logic pulser being used. Typically, a TTL gate has an output resistance between 12Q (low state) and 70Q (high state). When a logic pulser drives the output ofa NAND gate, the equivalent circuit appears as shown in Fig. 5.4b. Because of the low output impedance (2Q) of the logic pulser, most of the voltage pulse appears across the load (12 to 70Q). Therefore, the output is briefly driven into the opposite voltage state. Number Systems and Codes 2Q 2Q r +5~_f1__ or +5~--u-(a) (b) +5V 2Q Test +5~_f1__ Short to 2Q supply Test node node or rt to und +5~--u-- = (c) = (d) (a) Thevenin equivalent of logic pulser, (b) Logic pulser driving NAND-gate output, (c) Node stuck in high state Testing Any Node You can use a logic pulser to drive any node in a circuit, whether input or output. Almost always, the load impedance of the node being driven is larger than the output impedance of the logic pulser. For this reason, the logic pulser can usually change the state of any node in a logic circuit. Also, the pulse width is kept very short (fractions of a microsecond) to avoid damaging the circuit being tested. (Note: Power dissipation is what damages ICs. A brief voltage pulse produces only a small power dissipation.) Stuck Nodes When is a logic pulser unable to change the state of a node? When the test node is shorted to ground or to the supply voltage. For instance, Fig. 5.4c shows the test node shorted to ground. In this case, all the voltage pulse is dropped across the internal impedance of the logic pulser; therefore the test node is stuck at OV, the low state. On the other hand, the test node may be shorted to the supply voltage as shown in Fig. 5.4d. Most power supplies are regulated and have impedances in fractions of 1Q. For this reason, most of the voltage pulse is again dropped across the output impedance of the logic pulser, which means that the test node is stuck at +5V. Finding Stuck Nodes If a circuit is faulty, you can use a logic pulser and logic probe to locate stuck nodes. Here's how. Touch both the logic pulser and the logic probe to a node as shown in Fig. 5.3. If the node is stuck in either state, the logic pulser will be unable to change the state. So, if the logic probe does not blink, you have a stuck node. Then, you can look for solder bridges on any

trace connected to the stuck node, or possibly replace the IC having the stuck node. Digital Principles and Applications 5.10 ERROR DETECTION AND CORRECTION Error Detection and Correction (EDAC) techniques are used to ensure that data is correct and has not been corrupted, either by hardware failures or by noise occurring during transmission or a data read operation from memory. There are many different error correction codes in existence. The reason for different codes being used in different applications has to do with the historical development of data storage, the types of data errors occurring, and the overhead associated with each of the error-detection techniques. We discuss some of the popular techniques here with details of Hamming code. Parity Code We have discussed parity generation and checking in detail in Section 4.8. When a word is written into memory, each parity bit is generated from the data bits of the byte it is associated with. This is done by a tree of exclusive-OR gates. When the word is read back from the memory, the same parity computation is done on the data bits read from the memory, and the result is compared to the parity bits that were read. Any computed parity bit that does not match the stored parity bit indicates that there was at least one error in that byte (or in the parity bit itself). However, parity can only detect an odd number of errors. If even number of errors occur, the computed parity will match the read parity, so the error will go undetected. Since memory errors are rare if the system is operating correctly, the vast majority of errors will be single-bit errors, and will be detected. Unfortunately, while parity allows for the detection of single-bit errors, it does not provide a means of determining which bit is in error, which would be necessary to correct the error. Thus the data needs to be read again if an error is detected. Error Correction Code (ECC) is an extension of the parity concept. Checksum Code This is a kind of error detection code used for checking a large block of data. The checksum bits are generated by summing all the codes of a message and are stored with data. Usually the block of data summed is of length 512 or 1024 and the checksum results are stored in 32 bits that allow overflow. When data is read, the summing operation is again done and checksum bits generated are matched with the stored one, If they are unequal, then an error is detected. Obviously, it can fool the detection system if error occurring at one place is compensated by the other. Cycle Redundancy Code (CRC) CRC code is a more robust error checking algorithm than the previous two. The code is generated in the following manner. Take a binary message and convert it to a polynomial, then divide it by another predefined polynomial called the key. The remainder from this division is the CRC. This is stored with the message. Upon reading the data, memory controller does the same operation, i.e. divides the message by the same key and compares with CRC stored. If they differ, then the data has been wrongly read or stored. Not all keys are equally good. The longer the key, the better is the error checking. On the other hand, the calculations with long keys can get quite complex. Two of the polynomials commonly used are: CRC-16 CRC-32 + x15 + x2+ 1 = X32 + X26 + X23 + X22 + X16 + X12 + XJ 1 + X10 + Xg + X7 + X5 + X4 + X2 + x + 1 =x 16 Usually, series of exclusive-OR gates are used to generate CRC code. We shall see in the next chapter that t~e sum term arising out of addition is essentially an exclusive-OR operation. Number Systems and Codes Hamming Code Introduced in 1950 by R W Hamming, this scheme allows one bit in the word to be corrected, but is unable to correct events where more than one bit in the word is in error. These multi-bit errors can only be detected, not corrected, and therefore will cause a system to malfunction. Hamming code uses parity bits discussed before but in a different way. For n number of data bits, if number of parity bits required here ism, then 2m;:::m+n+ 1 In the memory word, (i) all bit positions that are of the form i are used as parity bits (like 1, 2, 4, 8, 16, 32 ...) and (ii) the remaining positions are used as data bits (like 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 17, 18 ...) Thus code will be in the form of Pl P2 D3 P4D5 D6D7 P8 D9 DlO Dll ... where Pl, P2, P4, P8 ... are parity bits and D3, D5, D6, D7 ... are data bits. We discuss Hamming code generation with an example. Consider the 7-bit data to be coded is O110101. This requires 4 parity bits in position 1, 2, 4 and 8 so that Hamming coded data becomes 11-bit long. To calculate the value of PI, we check parity of zeroth binary locations of data bits. This is shown in 3rd row of Fig. 5.5 for this example. Zeroth locations are the places where address ends with a 1. These are D3, D5, D9 and D 11 for 7-bit data. Since we have total odd number of 1s in these 4 positions P 1 = I. This is calculated as done in case of parity generation (refer to Section 4.8) by series of exclusive-OR gates through the equation Pl= D3 EB D5 EB D9 EB Dll Similarly for P2, we check locations where we have 1 in address of the 1st bit, i.e. D3, D6, D7, D 10 and D 11. Since there are even number of 1s, P2 = 1. Proceeding in similar manner and examining parity of 2nd and 3rd position, we get P4 = 0 and P8 = 0. 0001 0010 OOll 0100 0101 0110 01 Il Pl P2 D3 P4 D5 D6 D7 Data word (without parity) Pl P2 P4 PS 1 Data with parity 1 0 0 0 I 1 1 0 0 0 0 0 1 I 0 0 0 0 0 1 .. 1 1 1000 1001 1010 1011 PS D9 DlO Dll 1 0 1 1 I 0 0 1 0 I 0 0 I 1 I Calculation of Parity Bits Next we discuss how error in a Hamming coded data is detected and if it is in single bit, how it is corrected. We continue with the previous example and consider that the data is incorrectly read in position D 11 so that 11-bit coded data is 10001100100. Figure 5.6 describes the detection mechanism. First of all, we check the parity of zeroth position and find it to be even. Since Pl = 1, the parity check fails and this is equivalent to generating a parity bit at the output (last column) following the equation Parity Pl check bit= D3 EB D5 EB D9 EB Dll EB Pl This is similar to parity checker in Section 4.8. Note that, in addition to data bits, we have also included the corresponding parity bit to the input of exclusive-OR gate tree. Proceeding similarly for other positions, we Digital Principles and Applications find that except for P4 all other parity checks fail. Note that, even a single failure detects an error. However, to correct the error, we use the output of last column 1011 (in the order PS P4 P2 Pl) and find its decimal equivalent which is 11. So the data of location 11, which is D 11 needs to be corrected. Received data word Pl P2 P4 PS Pl P2 D3 P4 I I 0 0 0 0 0 0 D5 D6 D7 PS D9 Dl0 Dll I 1 0 0 0 0 0 I I 0 0 0 0 0 0 0 1 1 0 1 I 0 1 Parity check Parity bit Fail Fail Pass Fail 1 1 0 1 Error detection and correction Note that, this method detects error in more than one position unlike.the first method but overhead is more. In simple parity method, we add I additional bit for 7-bit data whreas it is 4 in this method. Also note, by further increasing this overhead, error in more than one position can also be corrected. However, more than one-bit error is unlikely for memory read. With overhead for one-bit correction, ifthere occurs error in more than one-bit positions, then the data needs to be read once again from the memory. 18. 19. 20. 21. Can parity code detect even number of errors? What is the full form ofCRC? What is the advantage of Hamming code'? Whatis error detection-correction overhead? PROBLEM SOLVING WITH MUti8PlEMETH00S Add two gray coded numbers O100 and O111 and express the result in gray code. Solution Since gray coded numbers are not suitable for arithmetic operations, we h.ave to co;nvert the numbers to some Other form, perform the addition and then convert the resu!rto gray £Ode. \Ve first show how it Can be done through lookup tables. It would require storage oflargeJook:µp tables,ifthenumbers.are large in value. Next, we show the converter-based approach which only needs the implementation ofconversion equations. In Method-1, ·.· we take help of first two columns of Table 5.9 ~~dconvert these two numb.~rs t? ~ecimal, add the decimal numbers. and then again use the table to; get corresponding gray coded number: This is shown in Fig. 5.7a. · In Method-2, we take help oflast two c.olU1U11s of Table 5,,9 ~d convert th!;lse two numbers.to 1:>inary., perform binary addition and. then again use the table. to get corresponding gray coded number. This is shown in Fig. 5.7b. ·'" Method-3,.. we take help. of gray to. binary. conversion.relati?n sho;vn inFig. 5.2ba11d pot1yert these two numbers to binary, perform binary addition and then use binary to gray conversion relation' shown in Fig. 5.2a to get corresponding gray coded number. This is shown in Fig. 5.7c. Number Systems and Codes UsingTable.5.9 Gray 0100 Olll Binary Binary 0111 +0101 1100 1100 Gray 1010 (b) Addition using Method-2 From Fig. 5.2b Gra)'to Biliary Conversion: GfG2G1Go'" 0100: ForG]G2G1Go =0110: B3 =63 lJ3=0 B3=0 B2=B3$G2 B2=0$= l B2=0$ l= B1 B1 B1 @G1 =l $ 0 =1 = 1 @1 =O Bo=B1 $Go Binary Bo= l $0= I 0111 +0101 Bo= 0 EB 1 = I 1100 .From .Fig, 5.2a Binarytl) Gray C?t1V(:rsio11: ForB3 l12 B! lfo = 1100: (73=B3 B3 = 1 .B2= 1 $=0 G1=B2E!3B1 B1 = 1 $0= 1 Go.= Bo Bo= 0 = 0 Gray 1010 (c}Addition using.Methodc3 of To c?hverffrom ~in~ to decimal nµInbers, add the "'.eight each bi.t position (1, 2, 4, s, ...)when .there is a.1 in that po~ition. With fractions, the binary weights are _,.,and so on. To convert from decimal ~ do~ble da~ble for integers and the multiply-bY:.2method for fractions. to Th! b~se.of ~ n~ber systein equats the number of digits it uses, The decimal number systeIU has base oUO,while the binary number system has a base of 2. The octal number system has a base of8. Auseful model for counting is the octalodometer•When a display wheel turns from 7back to 0, it sends a carry to the next-higher wheel. Hexadecimal numbers have a base of 16. The model for counting is. the hexadecimalodometer, whose wheels reset and carry beyond F. He)lt~~eciIUaln~bers are ~asytoconvert mentallyinto thefrb' equivalents. For this reason, people prefetusinghexadecimalnunibers because they aremuchsharter the corresponding binary numbers. The ASCII code is an alphanumeric code widely used for transferring data into cu,~,v~u ~ 7-bi.t 5ode i~ used to. represe11t alphabet letters, numpers, and other symbols. The excess-3 the Gray code are two other codes that are used. A logic pulser can temporarily change the state of a node under test.If the original state is low, the logic pulser drives the node briefly into the high state. If the state is high, the logic pulser drives the node briefly bin.iry, J,{J, a Digital Principles and Applications into the low state. The output impedance of a logic pulser is so low that it can drive almo$t. any normal node in a logic circuit When a node is shorted to ground or to the supply voltage, thelogic pulseds unable to change the voltage level; this is a confirmation of the shorted condition. Parity code, Checksmn code, and CRC code have been discussed for error detectioncodeand Banµning code for error detection and correction. These techniques are used to ensure that datais correct and has not been corrupted, either by hardware failures or by noise occurring during transmission or a data read operation from memory. • base The number of digits or basic symbols in " hexadecimal Refers to number system with a a number system. The decimal system has a base of 10 because it uses 10 digits. Binary has a base of 2, octal a base of 8, and hexadecimal a base of 16. binary Refers to a number system with a base of 2, that is, containing two digits. bit An abbreviated fom1 of binary digit. Instead of saying that 10110 has five binary digits, we can say that it has 5 bits. byte A binary number with 8 bits. checksum code A error detection code generating sum of a block of data. CRC code Cyclic Redundancy Code is a polynomial key based error detection code. digit A basic symbol used in a number system. The decimal system has 10 digits, 0 through 9. error detection and correction A method of detection of e1Tor in a group of bits and correction of the same. hamming code A parity bit based error detection and correction code. base of 16. The hexadecimal system has digits 0 through 9, followed by A through F. logic pulser A troubleshooting device that generates brief voltage pulses. The typical logic pulser has a push-button switch that produces a single pulse for each closure. More advanced logic pulsers can generate a pulse train with a specified number of pulses. nibble An binary number with 4 bits. octal Refers to a number system with a base of 8, that is, one that uses 8 digits. Normally, these are 0, 1, 2, 3, 4, 5, 6, and 7. parity code An error detection code using one additional parity bit. weight Refers to the decimal value of each digit position of a number. For decimal numbers, the weights are 1, 10, 100, 1000, . .. , working from the decimal point to the left. For binary numbers the weights are 1, 2, 4, 8, ... to the left of the binary point. With octal numbers, the weights become l, 8, 64, ... to the left of the octal point. • • • • • • • • 5.1 What is the binary number that follows 01101111? 5.2 How many bits are there in 2 bytes? • • " " • 5.3 How many nibbles are there in each of these: a. 1001 b. 11110000 Number Systems and Codes C. 110011110000 d. 1111000011001001 5.4 Give the decimal equivalents for each of the following binary numbers: a. 110101 b. 11001.011 5.5 Convert the following binary numbers to their decimal equivalents: a. 1011 1100 b. 11111111 5.6 What is the decimal equivalent of 1000 1100 1011 0011? 5. 7 A computer has 128K of memory. How many bytes does this represent? 5.8 Conve11 the following decimal numbers to binary numbers: 24, 65, and 106. 5.9 What binary number does decimal 268 stand for? 5.10 Convert decimal 108.364 to a binary number. 5.11 Calculate the binary equivalent for 5280. 5.12 Convert the following octal numbers to decimal equivalents: a. 65 b. 216 C. 4073 5.13 What is the decimal equivalent of octal 325.736? 5.14 Convert these decimal numbers to octal numbers: a. 4096 b. 65535 5.15 What is the octal equivalent of decimal 324.987? 5 .16 Convert the following octal numbers to binary numbers: 34,567, 4673. 5 .17 Convert the following binary numbers to octal numbers: a. 10101111 b. 1101.0110111 C. 1010011.101101 5.18 What are the hexadecimal numbers that follow each of these: a. ABCD b. 7F3F c. BEEF 5.19 Convert the following hexadecimal numbers to binary numbers: a. E5 b. B4D C. 7AF4 5.20 Convert these binary numbers into hexadecimal numbers: b. 0011 0111 a. 1000 1100 c. 1111 0101 0110 5.21 Convert hexadecimal 2F59 to its decimal equivalent. 5.22 What is the hexadecimal equivalent of decimal 62359? 5.23 Give the value of Y3Y2Y1Yo in Fig. 5.8 for each of these: a. All switches are open b. Switch 4 is closed c. Switch A is closed d. Switch F is closed 5.24 A computer has the following hexadecimal contents stored at the addresses shown: Hexadecimal contents Address 2000 D5 2001 AA 2002 96 2003 DE 2004 AA 2005 EB What are the binary contents at each address? 5.25 Give the ASCII code for each of these: a. 7 b. W C. f d. y 5.26 Suppose that you type LIST with an ASCII keyboard. What is the binary output as you strike each letter. Digital Principles and Applications +5V ·~ 0 1 - 2- •,-1\/1/\ •>-W. 3 4 ~ 5 6 ~ .~ ~ .~ ,L.J\AA VVv ·- 7 . 8 9 'LA.., ..f'v ~"v ~\ ~./'v A~ B C ~A ~\ ~v D- I All resistors F_ E Ml A 'V V are 101& I \l - .~ I y7430 y7430 y7430 97430 5.27 In Example 5.15, a computer sends the word HELLO to another computer. The characters are coded in ASCII with an odd-parity bit. Here is how the word is stored in the memory of the receiving computer: Address Alphanumeric Hexadecimal contents 2000 2001 2002 2003 2004 H E L L 0 cs 45 4C 4C 4F The transmitting computer then sends the word GOODBYE. Show how this word is stored in the receiving computer. Use a starting address of 2000 and include a parity bit. 5.28 Express decimal 5280 in excess-3 code. 5.29 Here is an excess-3 number: 0110 1001 1100 0111 What is the decimal equivalent? 5.30 What is the Gray code for decimal 8? 5.31 Convert Gray number 1110 to its BCD equivalent. 5.32 Figure 5.9 shows the decimal-to-BCD encoder discussed in Sec. 4.6. Answer the following questions: a. If all switches are open and the logic pulser is inactive, what voltage level does the logic probe indicate? b. If switch 6 is closed and the logic pulser is inactive, what does the logic probe indicate? _ _ _ _ _ _ _ _ _ _ _ _ _ _N_u_m_b_er_S_y_st_em_s_a_n_d_C_o_de_s_ _ _ _ _ _ _ _ _ _ _ _ _ _ c. If all switches are open and the logic pulser is activated, what does the logic probe do? 5.33 The push-button switch of the logic pulser shown in Fig. 5.9 is pressed. Suppose that the logic probe is initially dark and remains dark. Indicate which of the following are possible sources of trouble: a. 74147 defective b. Pin 9 shorted to ground c. Pin 9 shorted to +5 V d. Pin 10 shorted to ground 5.34 The instruction register shown in Fig. 5.10 on the next page is a logic circuit that stores a 16-bit number, /1 5 ... Io. The first 4 bits, /is ... I12, are decoded by a 4 to 16-line decoder. Determine whether the logic probe indicates low, high, or blink for each of these conditions: a. I1s ... /12 = 0000 and logic pulser inactive b. /is ... / 12 = 1000 and logic pulser inactive c. I1s ... / 12 = 1000 and logic pulser active d. I1s ... 112 = 1111 and logic pulser active 5.35 The logic pulser and logic probe shown in Fig. 5.10 are used to check the pins of the 7404 for stuck states. Suppose pin 8 is stuck in the high state. Indicate which of the following are possible sources of trouble: a. No supply voltage anywhere in circuit b. Pin I ofIC2 shorted to ground c. Pin 2 ofIC4 shorted to the supply voltage d. Pin 3 ofIC5 shorted to ground e. Pin 4 ofIC8 shorted to the supply voltage 5 .36 Find Hamming code of data 11001. 5.37 Find Hamming code of data 1000111. 5.38 If an error occurs in the 3rd data bit, how will it be corrected for data of problem 5.37? 5.39 How many parity bits are needed to Hamming code (a) 16-bit data and (b) 24-bit data. +SV All resistors are 1 kQ 1-1-.. 2-1-.. 3-1-.. 4-1-.. 5-1-.. 6-1-.. 7-1-.. 8-1-.. 9-1-.. ~ 16 11 xl Vee 12 X 74147 2 13 A X3 2 3 4 5 10 X4 X5 B x6 C X1 Xg D X 9 GND - 8 - 14 6 7 9 Digital Principles and Applications Instruction register I15 113 114 lo I12 ·~ ·~ ·~ R7 2 4 6 7404 8 I '' 6 4 ICI 9 IO 8 12 13 I ' 6 3 4 IC2 9 10 12 8 13 I 2 6 3 4 9 - IC3 10 12 C (b) (a) Full-adder, (b) Karnaugh map of Table 6.3 When you examine each entry in Table 6.3, you can see that a full-adder performs binary addition on 3 bits. From this truth table we get Karnaugh map as shown in Fig. 6.4b that gives following logic equations, CARRY=AB+BC+AC and SUM=A EBB$ C. A general representation of full-adder which adds i-th bit A; and B; of two numbers A and Band takes carry from (i-l)th bit could be C; = A;B; + B;C;-1 + A;CH or C; = A;B; + (A; +B;)C;-1 and S; =A;$ B; $ C;_ 1 where, C; and S; are carry and sum bits generated from the fall adder. The second representation of C; has an interesting meaning. The first term gives, if both A; and Bi are 1 then C; = 1. The second term gives if any of A; or B; is 1 and ifthere is carry from previous stage, i.e. C;_ 1 = 1 then also C; = 1. That this is the case, we can verify from full adder truth table and this understanding is useful in design of fast adder in Section 6.9. Digital Principles and Applications Controlled Inverter Figure 6.5 shows a controlled inverter. When INVERT is low, it transmits the 8-bit input to the output; when INVERT is high, it transmits the l's complement. For instance, if the input number is A1---Ao=OllO 1110 Controlled inverter a low INVERT produces Y1···Yo=OI10 1110 But a high INVERT results in Y1 ... Yo= 1001 0001 The controlled inverter is important because it is a step in the right direction. During a subtraction, we first need to take the 2's complement of the subtrahend. Then we can add the complemented subtrahend to obtain the answer. With a controlled inverter, we can produce the l's complement. There is an easy way to get the 2 's complement, discussed in the next section. So, we now have all the building blocks: half-adder, full-adder, and controlled inverter. 13. What are the it1pt1tsJndotltptitsof iihalf-ad~er? .... -------lJNVB V3 =+5 Vdc (c) Bistable circuit switch the flip-flop from one stable state to the other. Two 2-input NOR gates are connected in Fig. 8.3a to fom1 a flip-flop. Notice that if the two inputs labeled R and Sare ignored, this circuit will function exactly as the one shown in Fig. 8.. 2a. 5 R S v;;-J.. v3 . NORA NORB NORB (a) (b) NOR-gate flip-flop This circuit is redrawn in a more conventional form in Fig. 8.3b. The flip-flop actually has two outputs, defined in more general terms as Q and Q. It should be clear that regardless of the value of Q, its complement is Q. There are two inputs to the flip-flop defined as R and S. The input/output possibilities for this RS flipflop are summarized in the truth table in Fig. 8.4. To aid in understanding the operation of this circuit, recall that an H = I at any input of a NOR gate forces its output to an L = 0. I. The first input condition in the truth table is R = 0 and S = 0. Since a O at the input of a NOR gate has no effect on its output, the flip-flop simply remains in its present state; that is, Q remains unchanged. 2. The second input condition R = 0 and S = I forces the output of NOR gate B low. Both inputs to NOR gate A are now low, and the NOR-gate output must be high. Thus a I at the S input is said to SET the flip-flop, and it switches to the stable state where Q = 1. Flip-Flops 3. The third input condition is R = I and S = 0. This Action R s Q condition forces the output of NOR gate A low, and No change 0 since both inputs to NOR gate B are now low, the SET 0 output must be high. Thus a 1 at the R input is said to RESET the flip-flop-'-and it switches to the stable RESET 0 0 state where Q = 0 (or Q = 1). Forbidden ? 4. The last input condition in the table, R = 1 and S = 1, is forbidden, as it forces the outputs of both NOR Truth table for a NORgates_!o the low state. In other words, both Q = 0 gate RS flip-flop and Q = 0 at the same time! But this violates the basic definition of a flip-flop that requires Q to be the complement of Q, and so it is generally agreed never to impose this input condition. Incidentally, if this condition is for some reason, imposed and the next input is R = 0, S = 0 then the resulting state Q depends on propagation delays of two NOR gates. If delay of gate A is less, i.e. it acts faster, then Q = 1 else it is 0. Such dependence makes the job of a design engineer difficult, as any replacement of a NOR gate will make Q unpredictable. That's why R = 1, S = 1 is forbidden and truth table entry is ? . It is also important to remember that TTL gate inputs are quite noise-sensitive and therefore should never be left unconnected (floating). Each input must be connected either to the output of a prior circuit, or if unused, to GND or + VccUse the pinout diagram for a 54/7427 triple 3-input NOR gate and show how to connect a simple RS flip-flop. Solution One possible arrangement is shown in Fig. 8.5. Notice that pins 3 and 4 are tied together, as are pins 10 and 11; thus no input leads are left unconnected and the two gates simply function as 2-input gates. The third NOR gate is not used. (It can be a spare or can be used elsewhere.) 54/7427 ... __ ., +Vee 2 13 R Q 54/7427 Digital Principles and Applications The standard logic symbols for an RS flip-flop are shown in Fig. 8.6 along with its truth table. The truth table is necessary since it describes exactly how the flip-flop functions. n: !l Q s R 0 0 I 0 0 - IEEE symbol Q Last state 0 ? (Forbidden) Logic symbol (a) (b) Truth table RS flip-flop NANO-Gate latch A slightly different latch can be constructed by using NAND gates as shown in Fig. 8.7. The truth table for this NAND-gate latch is different from that for the NOR-gate latch. We will call this latch an RS flip-flop. To understand how this circuit functions, recall that a low on any input to a NAND gate will force its output high. Thus a low on the S input will set the latch (Q = 1 and Q = 0). A low on the R input will reset it (Q = 0). If both R and S are high, the flip-flop will rem~n in its previous state. Setting both Rand S low simultaneously is forbidden since this forces both Q and Q high. :n: n s R s Last state IEEE symbol R (a) NAND gate latch Logic symbol (b) Q 0 0 0 0 0 ? (Forbidden) (c) Truth table is flip-flop Show how to convert the RS flip-flop in Fig. 8.7 into an RS flip-flop. Solution By placing an inverter at each input as shown in Fig. 8.8, the 2 inputs are now R and S, and the resulting circuit behaves exactly as the RS flip-flop in Fig. 8.6. A single 54/7400 (quad 2-input NAND gate) is used. Simple latches as discussed in this section can be constructed from NAND or NOR gates or obtained as mediumscale integrated circuits (MSI). For instance, the 74LS279 is a quad RS latch. The pinout and truth table for this circuit are given in Fig. 8.9. Study the truth table carefully, and you will see that the latches behave exactly like the RS flip-flop discussed above. Flip-Flops Q s n Q R (a) 54/7400 R s 0 0 Last state 0 0 Q 0 ? (Forbidden) (c) (b) Logic symbol An RS flip-flop (latch) R Vee Q 16 0 0 0 X X 0 0 0 ? Forbidden 0 ? Forbidden R :S\ S2 R S1 (a) Pinout 74S279A GND X=Don'tcare (b) Truth table Quad SET-RESET latch The NOR-gate flip-flop in Fig. 8.3 is seen to be an active-high circuit because an H = I at either the S or R input is required to change the output Q. On the other hand, the NAND-gate flip-flop in Fig. 8.7 can be considered an active-low circuit because an L = 0 at either input is required to change Q. The NAND gates in Fig. 8. 7 can be changed to bubbled-input OR gates as shown in Fig. 8.10. This circuit is equivalent to the NAND-gate latch in Fig. 8.7 and functions in exactly the same way. However, the bubbled inputs more clearly express circuit operation. Rs flip-flop Bubbled OR-gate equivalent of Fig. 8.7 Digital Principles and Applications 8.2 GATED fUP-fLOPS Two different methods for constructing an RS flip-flop were discussed in Sec. 8.1. The NOR-gate realization in Fig. 8.3b is an exact equivalent of the NAND-gate realization in Fig. 8.8a, and they both have the exact same symbol and truth table as given in Fig. 8.6. Both of these RS flip-flops, or latches, are said to be transE_arent; that is, any change in input infom1ation at R or S is transmitted immediately to the output at Q and Q according to the truth table. Clocked RS Flip-Flops The addition of two AND gates at the R and S inputs as shown in Fig. 8.1 I· will result in a flip-flop that can be enabled or disabled. When the ENABLE input is low, the AND gate outputs must both be low and changes in neither R nor Swill have any effect on the flip-flop output Q. The latch is said to be disabled. When the ENABLE input is high, information at the R and S inputs will be transmitted directly to the outputs. The latch is said to be enabled. The output will change in response to input changes as long as the ENABLE is high. When the ENABLE input goes low, the output will retain the information that was present on the input when the high-to-low transition took place. In this fashion, it is possible to strobe or clock the flip-flop in order to store information (set it or reset it) at any time, and then hold the stored information for any desired period oftime. This flip-flop is called a gated or clocked RS jlip:flop. The proper symbol and truth table are given in Fig. 8.11 b. Notice that there are now three inputs--R, S, and the ENABLE or CLOCK input, labeled EN. Notice also that the truth-table output is not simply Q, but Q11 + 1• This is because we must consider two different instants in time: the time before the ENABLE goes low Q11 and the time just after ENABLE goes low Q11 + 1. When EN= 0, the flip-flop is disabled and R and Shave no effect; thus the truth table entry for R and Sis X (don't care). Explain the meaning of Q11 the truth table in Fig. 8.11 b. EN s U ENABLE R (a) Logic diagram o s R 0 0 0 Q11+I Q11 (no change) 0 0 Q ? (Illegal) 0 X X Q11 (no change) (b) IEEE symbol and truth table Clocked RS flip-flop Solution For the nip-flop to operate properly, there must be a PT on the EN input. While EN is high, the information on R and S causes the latch to set or reset. Then when EN transitions back to low, this infom1ation is retained in the latch, When this NT occurred, both Rand S inputs ;were low (0), and thus there was no change of state. In other words, the value of Q at time n + l is the same as it was at time n. Remember that time n occurs just before the NT on EN, and time n + I, occurs just after the transition. The logic diagrams shown in Fig. 8. l2a and b illustrate two different methods for realizing a clockRS flip-flop. Both realizations are widely used in medium- and large-scale integrated circuits, and you will find them easy to recognize. You might like to examine the .logic diagrams for a 54LS109 or a 54LS74, for instance. Flip-Flops s s EN EN R R (b) (a) Two different realizations for a clocked RS flip-flop Figure 8.13 shows the input wavefom1s R, S, and EN applied to a clocked RS !lip-flop. Explain the output waveform Q. ti f3 14 t5 f2 16 t7 Is Time - - - - - - - - - - - - - -! -! - - ' I EN 0l _l-i .---l I I R 0--..,..,-~ : ~--~h'. . ___ ... - ·n.....__ __ I I s 0 Q I 0 .____.,__,rL_n.....______ Solution Between t2 and t3 both R and S change states, but since EN is low, the flip-flop is still disabled and Q remains at I. Between t3 and 16, the flip-flop will respond to any change in Rand S since EN is high. Thus at t3 Q goes low, and at t4 it goes back high. No change occurs atts. At t6 the value Q = 1 is latched and no changes in Q occur between ft, and t1 even though both R and S change. Between t 7, and ts no change in Q occurs since both Rand Sare low. Initially, the flip-flop is reset (Q 0). At time t 1 EN goes high; the flip-flop is now enabled, and it is immediately set (Q = l) since R = 0 and S = 1. At time t2 EN goes low and the flip-flop is disabled and latches in the stable state Q = 1. Clocked D Flip-Flops The RS flip-flop has two data inputs, R and S. To store a high bit, you need a high S; to store a low bit, you need a high R. Generation of two signals to drive a flip-flop is a disadvantage in many applications. Furthermore, the forbidden condition of both R and S high may occur inadvertently. This has led to the D flip-flop, a circuit that needs only a single data input. Figure 8.14 shows a simple way to build a D (Data) flip-flop. This flip-flop is disabled when EN is low, but is transparent when EN is high. The action of the circuit is straightforward, as follows. When EN is low, both AND gates are disabled; therefore, D can change value without affecting the value of Q. On the other hand, Digital Principles and Applications when EN is high, both AND gates are enabled. In this case, Q is forced to equal the value of D. When EN again goes low, Q retains or stores the last value of D. There are many ways to design D flip-flops. In general, a D flip-flop is a bistable circuit whose D input is transferred to the output when EN is high. Figure 8.15 shows the logic symbols A D Flip-flop used for any type of D flip-flop. In this section we're talking about the kind of D flip-flop in which Q can follow the value of D while EN is high. In other words, if the data bit changes while EN is high, the last value of D before EN return low is the value of D that is stored. This kind of D flip-flop is often called aD latch. Figure 8.15b shows the truth table for a D latch. While (EN) is low, Dis a don't care (X); Q will remain latched in its last state. When EN is high, Q takes on the last value of D. If D is changing while EN is high, it is the last value of D that is stored. =fi-o~ [J-Q IEEE symbol EN D 0 ~ Qn+I X Q11 (last state) 0 0 Logic symbol (a) D flip-flop logic symbol (b) Truth table D Flip-flop logic symbol The idea of data storage is illustrated in Fig. 8.16. Four D latches are driven by the same clock signal. When the clock goes high, input data is loaded into the flip-flops and appears at the output. Then when the clock goes low, the output retains the data. For instance, suppose that the data input is D3D2D1Do = 0111 When the clock goes high, this word is loaded into the D latches, resulting in an output of D3 D2 Di Do Clock DEN DEN DEN DEN Q Q Q Q Q Q Q Q Q3 Q2 QI Qo Storing a 4-hit word Q3Q2Q1Qo = 0111 After the clock goes low, the output data is retained, or stored. As long as the clock is low, the D values can change without affecting the Q values. The 7475 in Fig. 8.17 is a TTL MSI circuit that contains four D latches; it's called a quad bistable latch. The 7475 is ideal for handling 4-bit nibbles of data. If more than one 7475 is used; words of any length can be stored. 4. What does an entry X mean ina flip-flop truth table? 5. What could you do to disable the flip-flop in Fig. 8.11? 6. \Vhich flip-flop is easier to use, the RS of the D, as a clocked or gated latch to store data? Flip-Flops +Vee 7 6 12 - 3 D2 D3 Vee D1 EN 1 7475 GND 2 Do EN2 Q3 Q3 Q2 Q2 QI QI Qo Qo 9 8 10 11 4 13 15 14 16 (a) D EN Q (b) 4-bit bistable latch: (a) Pinout, (b) Logic diagram (each latch) 8.3 EDGE-TRIGGERED RS f UP-flOPS The simple latch-type flip-flops presented in Sec. 8.1 are completely transparent; that is, the output Q immediately follows any change of state at the input (R, S, or D). The gated or clocked RS and D flip-flops in Sec. 8.2 might be considered semitransparent. That is, the output Q will change state immediately provided that the EN input is high. If any of these flip-flops are used in a synchronous system, care must be taken to ensure that all flip-flop inputs change state in synchronism with the clock. One way of resolving the problem for gated flip-flops is to allow changes in R, S, and D input levels only when EN is low (or require fixed levels at R, S, and D any time EN is high). At the very least, these are highly inconvenient restrictions, and at the worst they may in fact be impossible to realize. From the previous chapter, we know that virtually all digital systems operate in a synchronous mode. Thus the edge-triggeredflip-flop was developed to overcome these rather severe restrictions. Positive-Edge-Triggered RS Flip-flops In Fig. 8.18a, the clock (C) is applied to a positive pulse-forming circuit (discussed in Sec. 7.1). The PTs developed are then applied to a gated RS flip-flop. The result is a positive-edge-triggered RS flip-flop, with the IEEE symbol given in Fig. 8.18b. The small triangle inside the symbol (dynamic input indicator) indicates that Q can change state only with PTs of the clock (C). Each PT of the clock in Fig. 8.18c produces a very narrow PT that is applied to the AND gates. The AND gates are active only while the PT is high (perhaps 25 ns), and thus Q can change state only during this short time period. In t11is manner Q changes state in synchronism with the PTs of the clock. Principles and Applications fr: tl-Q (a) Logic diagram (b) IEEE symbol C C s R + 0 + 0 + t Action No change 0 RESET 0 SET ? Illegal (c) Truth table 0 Q,,+J PT ---J'--_,.______.,____...c__...,___ 10 t1 t, t3 t4 i I I Q,, Q----' (d) Positive-edge-triggered RS flip-flop Positive-edge-triggered RS flip-flop This flip-flop is easy to use in any synchronous system! Another way of expressing its behavior is to say the flip-flop is transparent only during PTs; it is not transparent for the remainder of the time. In other words, S and R inputs affect Q only while the positive pulse is high, and they need to be static only during this very short time. The truth table for the edge-triggered RS flip-flop is given in Fig. 8.18c. The small vertical arrows under C (clock) mean that changes of state (Q) occur according to the R and S levels, but only during PTs of the clock. Look at the wavefonns in Fig. 8.18d. Note that when Q changes state, it does so in exact synchronism with PTs of the clock C. Use the positive-edge-triggered RS flip-flop truth table to explain Q changes of state with time in Fig. 8.18d. Solution Here's what happens at each pointin time: Time t0: S = O, R = 0, no change in Q(Q remains O) Time t1: S = 1, R = O, Q changes from Oto l Time t2: S = O, R = l, Q resets to 0 Time t3: S = 1,R = 0, Q sets to 1 Time t4: S = O, R = 0, no change in Q(Q remains 1) 0 Notice that either R or S, or both, are allowed to change state at any time, whether C is high or low. The only time both R and S must be stable (unchanging) is during the short PTs of the dock. Negative-Edge-Triggered RS Flip-Flops The symbol in Fig. 8.19a is for a negative-edge-triggered RS flip-flop. The truth table in Fig. 8.19b shows that Q changes state according to the R and S inputs, but only during NTs of the clock. On the IEEE symbol, the small bubble on the clock input (C) means active-low. This bubble, along with the dynamic input indicator, n: Q C s R t 0 0 t t t (a) IEEE symbol Q11+ 0 Qll 1 0 Action l No change RESET SET Illegal 0 ') (b) Truth table Negative-edge-triggered RS flip-flop means negative-edge triggering. This flip-flop behaves exactly like the positive-edge-triggered RS flip-flop, except that changes inoutput Qare synchronized with NTs of the clock (C). Use the negative-edge-triggered RS flip-flop truth table to explain Q changes of state with time in Fig. 8.20. Solution Here's what happens at each point in time: Time to: S= 0, R = 0, no change in Q(Q remains 0) Time t1 : S = 1, R = 0, Q changes from Oto 1 Time t2: S = 0, R =l, Q resets to 0 Time t3: S=l, R =O, Q sets to 1 Time t4: S = O, R = O, no change in Q(Q remains 1) Notice that either R or S, or both, are allowed to change state at any time, whether C is high or low. The only time both R and S must be stable (unchanging) is during the short NTs of the clock. C PT to s ti t2 t3 t4 I I R Q 7. What does it mean to say that a flip-flop is transparent? 8. What is positive-edge triggering? 9. How does an RS latch differ from an edge-triggered RS flip-flop? 8.4 EDGE-TRIGGERED D fUP-flOPS Although the D latch is used for temporary storage in electronic instruments, an even more popular kind of D flip-flop is used in digital computers and systems. This kind of flip-flop samples the data bit at a unique point in time. Figure 8.21 shows a positive pulse-forming circuit at the input of a D latch. The narrow positive pulse (PT) enables the AND gates for an instant. The effect is to activate the AND gates during the PT of C, which is equivalent to sampling the value of D for an instant. At this unique point in time, D and its complement hit the flip-flop inputs, forcing Q to set or reset (unless Q already equals D). Again, this operation is called edge triggering because the flip-flop responds only when the clock is in transition between its two voltage states. The triggering in Fig. 8.21 occurs on the positive-going edge of the clock; this is why it's referred to as positive-edge triggering. Digital Principles and Applications C D 0 X Qn+I Q11 (last state) t O + (a) Circuit diagram 0 (c) Truth table Positive-edge-triggered D flip-flop The truth table in Fig. 8.21 b summarizes the action of a positive-edge-triggeredD flip-fl.op. When the clock is low, D is a don't care and Q is latched in its last state. On the leading edge of the clock (PT), designated by the up arrow, the data bit is loaded into the flip-flop and Q takes on the value of D. When power is first applied, flip-flops come up in random states. To get some computers started, an operator his required at the preset input. What is the resulting state of QJ · 8S EDGE-lRIGGEREDJK··fUP-fl.OPS Setting R = S = I with an edge-triggered RS flip-flop forces both Q and Q to the same logic level. This is an illegal condition, and it is not possible to predict the final state of Q. The JK flip-flop accounts for this illegal input, and is therefore a more versatile circuit. Among other things, flip-flops can be used to build counters. Counters can be used to count the number of PTs orNTs of a clock. For purposes of counting, theJK flip-flop is the ideal element to use. There are many commercially available edge-triggered JK flip-flops. Let's see how they function. Positive-Edge-Triggered JK Flip-Flops In Fig. 8.24, the pulse-forming box changes the clock into a series of positive pulses, and thus this circuit will be sensitive to PTs of the clock. The basic circuit is identical to the previous positive-edge-triggered RS flip-flop, with two important additions: 1. The Q output is connected back to the input of the lower AND gate. 2. The Q output is connected back to the input of the upper AND gate. This cross-coupling from outputs to inputs changes the RS flip-flop into a JK flip-flop. The previous S input is now labeled J, and the previous R input is labeled K. Here's how it works: 1. When J and Kare both low, both AND gates are disabled. Therefore, clock pulses have no effect. This first possibility is the initial entry in the truth table. As shown, when J and K are both Os, Q retains its last value. 2. When J is low and K is high, the upper gate is disabled, so there's no way to set the flip-flop. The only possibility is reset. When Q is high, the lower gate passes a RESET pulse as soon as the next positive @) _______________ D_ig_it_al_P_n_·nc_ip_le_s_a_n_d_A_pp_ll_·ca_t,_·on_s_ _ _ _ _ _ _ _ _ _ _ _ _ C J K + 0 0 t 0 0 + C t Action Qn+I Q11 (last state) No change 0 Q11 (toggle) RESET SET Toggle (b) Truth table (a) One way to implement a JKflip-flop A positive-edge-triggered JK flip-flop clock edge arrives. This forces Q to become low (the second entry in the truth table). Therefore, J = 0 and K = I means that the next PT of the clock resets the flip-flop (unless Q is already reset). 3. When J is high and K is low, the lower gate is disabled, so it's impossible to reset the flip-flop. But you can set the flip flop as follows. When Q is low, Q is high; therefore, the upper gate passes a SET pulse on the next positive clock edge. This drives Q into the high state (the third entry in the truth table). As you can see, J = 1 and K = 0 means that the next PT of the clock sets the flip-flop (unless Q is already high). 4. When J and Kare both high (notice that this is the forbidden state with an RS flip-flop), it's possible to set or reset the flip-flop. If Q is high, the lower gate passes a RESET pulse on the next PT. On the other hand, when Q is low, the upper gate passes a SET pulse on the next PT. Either way, Q changes to the complement of the last state (see the truth table). Therefore, J = I and K = I mean the flip-flop will toggle (switch to the opposite state) on the next positive clock edge. Propagation delay prevents the JK flip-flop from racing (toggling more than once during a positive dock edge). Here's why. In Fig. 8.24, the outputs change after the PT of the clock. By then, the new Q and Q values are too late to coincide with the PTs driving the AND gates. For instance, if tP = 20 ns, the outputs change approximately 20 ns after the leading edge of the clock. If the PTs are narrower than 20. ns, the returning Q and Q arrive too late to cause false triggering. Figure 8.25a shows a symbol for a JK flip-flop of any design. When you see this on a schematic diagram, remember that on the next PT of the clock: I. 2. 3. 4. J and K low: no change of Q. J low and K high: Q is reset low. J high and K low: Q is set high. J and K both high: Q toggles to opposite state. You can include OR gates in the design to accommodate PRESET and CLEAR as was done earlier. Figure 8.25b gives the symbol for a JK flip-flop with PR and CLR. Notice that it is negative-edge-triggered and requires a low PR to set it or a low CLR to reset it. £1-o Q =E_J--Q (a) Basic symbol Q (b) 74LS76A JK flip-flop ¥ . (c) 74LS73A symbols Q Q Flip-Flops Figure 8.25c is another commercially available JK flip-flop. It is negative-edge-triggered and requires a low CLR to reset it. The output Q reacts immediately to a PR or CLR signal. Both PR and CLR are asynchronous, and they override all other input signals. Toggle flip-flop, popularly known as T flip-flop has following input-output relation. When input T = 0, the output Q does not change its state. For T = 1, the output Q toggles its value. Derive T flip-flop from JK flip-flop. From Fig. 8.24b we find for input J = K = 0, tbe output Qn+ 1 = Q,,, i.e. output does not change its state. And forJ=K= l, tbe output Qn+l = Q,;, i.e. output toggles. Thus, ifwe tieJandKinputs of JKflip-flop together and make a common input T=J=K, the resulting circuit will behave as.Tflip-fl.op. Solution 12. What is the primary difference between aJKand an.RS flip-flop? 13. How could you change an edge-triggered RS flip-flop into an edge-triggered JK flip-flop? 8.6 fUP~fLOP TIMING Diodes and transistors cannot switch states immediately. It always takes a small amount of time to tum a diode on or off. Likewise, it takes time for a transistor to switch from saturation to cutoff, and vice versa. For bipolar diodes and transistors, the switching time is in the nanosecond region. Switching time is the main cause of propagation delay, designated fp. This represents the amount of time it takes for the output of a gate or flip-flop to change states after the input changes. For instance, if the data sheet of an edge-triggered D flip-flop lists Ip= 10 ns, it takes about 10 ns for Q to change states after D has been sampled by the clock edge. This propagation delay time has been used to construct the "pulse-fom1ing circuit" used with edge-triggered flip-flops. When flip-flops are used to construct counters, the propagation delay is often small enough to be ignored. Stray capacitance at the D input (plus other factors) makes it necessary for data bit D to be at the input before the clock edge arrives. The setup time fsetup is the minimum amount of time that the data bit must be present before the clock edge hits. For instance, if a D flip-flop has a setup time of 15 ns, the data bit to be stored must be at the D input at least 15 ns before the clock edge arrives; otherwise, the manufacturer does not guarantee correct sampling and storing. Furthermore, data bit D has to be held long enough for the internal transistors to switch states. Only after the transition is assured can we allow data bit D to change. Hold time thold is the minimum amount of time that data bit D must be present after the PT of the clock. For example, if tsetup = 15 ns and thold = 5 ns, the data bit has to be at the D input at least 15 ns before the clock edge arrives and held at least 5 ns after the clock PT. Typical waveforms for setting a 1 in a positive-edge-triggered flip-flop are shown in Fig. 8.26. Discuss the timing. Solution The lower line in Fig. 8.26 is tbe time line with..critical times marked on it. Prior to t 1, the data can be a 1 or a 0, or can be changing. This is shown by drawing lines for both high and low levels on D. From time ti to t2, the Digital Principles and Applications Serial _data ... · ..8 bits · · data mput ..·. . . ...· output (a) Serial in-serial out MSB LSB '------r----1 Parallel data outputs (b) Serial in-parallel out Parallel data inputs ,------A--, MSB Parallel data inputs ,------A--, MSB LSB Serial data output (c) Parallel in-serial out LSB OJ MSB LSB '------r----1 Parallel data outputs (d) Serial in-parallel out Shift register types Digital Principles and Applications · Serial in-parallel out-54/74164, 8 bits Parallel in-serial out-54/74165, 8 bits Parallel in-parallel out-54/74198, 8 bits We now need to consider the methods for shifting data in either a serial or parallel fashion ..Data shifting techniques and methods for constructing the four different types of registers are discussed in the following sections. 9.2 SERIAL IN-SERIAL OUT In this section we discuss how data is serially entered or exited from a shift register. The flip-flops used to construct registers are usually edge-triggered JK, SR or D types. We begin our discussion with shift registers made from D type flip-flops and then extend the idea to other types. Consider four D flip-flops connected as shown in Fig. 9.2a forming 4-bit shift register. A common clock provides trigger at its negative edge to all the flip-flops. As output of one D flip-flop is com1ected to input of the next at every Clock trigger data stored in one flip-flop is transferred to the next. For this circuit transfer takes place like this Q ~ R, R ~ S, S ~ T and serial data input is transferred to Q. Let us see how actual data transfer takes place by an example. Assume, all the flip-flops are initially cleared. Let a binary waveform, as shown along D of Fig. 9.2b be fed to serial data input of

the shift register. Corresponding Q, R, S, Tare also shown in the figure. At clock edge A, flip-flop Q has input O from serial data in D, flip-flop R has input O from output of Q, flip-flop S has input Ofrom output of Rand flip-flop Thas input Ofrom output of S. When clock triggers, these inputs get transferred to corresponding flip-flop outputs simultaneously so that QRST= 0000. Thus at clock trigger, values at DQRS is transferred to QRST. Time r l A B C D + + i t I I I 1 Clock~ Q s,riru data son,, { R data input J s l I 1 1 I I input K 1 0 s I I 1 1 1 I 1 f: 1 I I Q l ~ Q Clock LL :I :I K I 1 I Rl 0 SI J I 1 1 1 I I I 1 1 I I I 1 I I I l 1 I I 1 1 1 1 1 0 (a) (b) 4-bit serial input shift register I I 11 1 I :o Registers At clock edge B, serial data in= 0, i.e. DQRS = 0000. So after NT at B, QRST= 0000. Serial data becomes 1 in next clock cycle. At clock edge C, DQRS = 1000 and after NT QRST= 1000. Serial data goes to O in next clock cycle such that at clock edge D, DQRS = 0100 and after NT QRST = 0100. Example 9 .1 will give another illustration of such data transfer. A shift register made up of JK or SR flip-flops has non-inverting output Q of one flip-flop connected to J or S input of next flip-flop and inverting output Q' connected to Kor R input respectively. For the first flip-flop, between J and K (or Sand R) an inverter is connected and J (or S) input is treated as serial data in. Note that, in this configuration both JK and SR flip-flops effectively act like a D flip-flop. Show how a number 0100 is entered serially in a shift register shown in Fig. 9.2a using state table. Figure 93 presents the state table. The timing diagram corresponding to this is discussed in this section. Note how the data flow across the flip-flops is highlighted by arrow direction. Soh1tion Clock Serial input 0 0 2 0 0 Q R s T o~o~o~o o~o~o~o l~O~O~O O~l~O~O 0 0 l 0 Datil transfer through serial input in .a shift register Draw the waveforms to shift the number 0100 into the shift register shown in Fig. 9.3 on the next page. Solution The waveforms for this register will appear exactly as in Fig. 9.2 provided the waveform labeled K is eliminated and waveform J is labeled D. 4-bit serial input shift register At this point, we have developed the ideas for shifting data into a register in serial fonn; the serial data input can be classified as either JK or D, depending on the flip-flop type used to construct the register. Now, how about shifting data out of the register? Digital Principles and Applications Let's take another look at the register in Fig. 9.3a, and suppose that it has the 4-bit number QRST = 1010 stored in it. If a clock signal is applied, the waveforms shown in Fig. 9.4 will be generated. Here's what happens: A B C D Time--~--.--~-~1 I I I I I I I I Clock~ 0 Before Time A The register stores the number QRST= 1010. The LSB (a 0) appears at T. I I I I I I I I t + t t DI 0 At Time A The entire number is shifted one flip-flop to the right. AO is shifted into Q and the LSB is shifted out the right end and lost. The register holds the bits QRST= 0101, and the second LSB (a I) appears at T. I I I I !o Q~t] I I I I I I I I I I I I I I I I I R ~ Q At Time B The bits are all shifted one flip-flop to the right, a 0 shifts into Q, and the third LSB (a 0) appears at T. The register holds QRST= 0010. s~~ I I I I I I I J T~~ At Time C The bits are all shifted one flip-flop to the right, a 0 shifts into Q, and the MSB (a I) appears at T. The register holds QRST= 0001. At Time D The MSB is shifted out the right end and lost, a Oshifts into Q, and the register holds QRST = 0000. To summarize, we have caused the number stored in the register to appear at T (this is the register output) 1 bit at a time, beginning with the LSB, in a serial fashion, over a time period of four clock cycles. In other words, the data stored was shifted out of the register at flip-flop Tin a serial fashion. Thus, not only is this a serial-input shift register, it is also a serial-output shift register. It is important to realize that the stored number is shifted out of the right end of the register and lost after four clock times. Notice that the complement of the output data stream is also available at f. The pinout and logic diagram for a 74LS9 l shift register are shown in Fig. 9.5. This is an 8-bit TTL MSI chip. There are eight RS flip-flops connected to provide a serial input as well as a serial output. 14 Q 13 Q 12 11 10 GND A 9 CLK 8 B 5 6 7 74LS9l Vee 2 3 4 (a) DIP pinout A B (b) Logic diagram 74LS91 8-bit shift register Registers The clock input at each flip-flop is negative-edge-trigger-sensitive. However, since the applied clock signal is passed through an inverter, data will be shifted on the positive edges of the input clock pulses. The inverter connected between R and Son the first flip-flop means that this circuit functions as a D-type flip-flop. So, the input to the register is a single line on which the data to be shifted into the register appears serially. The data input is applied at either A (pin 10) or B (pin 12). Notice that a data level at A (or B) is complemented by the NAND gate and then applied to the R input of the first flip-flop. The same data level is complemented by the NAND gate and then complemented again by the inverter before it appears at the S input. So, a 1 at input A will set the first flip-flop (in other words, this 1 is shifted into the first flip-flop) on a positive clock transition. The NAND gate with inputs A and B simply provides a gating function for the input data stream if desired. If gating is not desired, simply connect pins 10 and 12 together and apply the input data stream to this connection. Examine the logic levels at the input of a 74LS91 and show how a 1 and then a Oare shifted into the register. Solution .The input logic and the firstflip-flop are redrawn in Fig. 9.6a, and a l is applied at the data input A. The ..R>illputis 0, .the Sjnput is 1, and the flip,,fiop y,,m clearly be set when the clock goes high. In other words, the 1 at the ~t!lfaputwill shift int() the ilip-fiop. fafig. ~.6b, a Ois applied at the datajnput A. The R input is 1, the S input is 0, and the flip-flop will be resetwhenthe~lock goes high. The inputO is thus shifted into the flip-flop. (b) Logic levels shown by arrows will reset the flip-flop l. Whalisthelai;gestdecim~l nuniber that can be stored (in binary form) in a 74LS91 register? Is a 74LS9hegister sensitive to PTs or to NTs? 9.3 SERIAL IN....PARALlEl OUT The second type of register mentioned in Sec. 9 .1 is one in which data is shifted in serially, but shifted out in parallel. In order to shift the data out in parallel, it is simply necessary to have all the data bits available as outputs at the same time. This is easily accomplished by connecting the output of each flip-flop to an output pin. For instance, an 8-bit shift register would have eight output lines-one for each flip-flop in the register. The basic configuration is shown in Fig. 9.lb. ~ _ _ _ _ _ _ _ _ _ _ _ _ _ _D_ig_it_al_P_n_·nc_ip_le_s_a_n_d_A_pp_lt_·ca_t,_·on_s_ _ _ _ _ _ _ _ _ _ _ _ _ The 54/74164 is an 8-bit serial input-parallel output shift register. The pinout and logic diagram for this device are given in Fig. 9.7. It is constructed by using RS flip-flops having clock inputs that are sensitive to NTs. A careful examination of the logic diagram in Fig. 9. 7b will reveal that this register is exactly like the 74LS9 l discussed in the previous section-with two exceptions: (1) the true side of each flip-flop is available as an output-thus all 8 bits of any number stored in the register are available simultaneously as an output (this is a parallel data output); and (2) each flip-flop has an asynchronous clear input. Thus a low level at the clear input to the chip (pin 9) is applied through an amplifier and will reset (clear) every flip-flop. Notice that this is an asynchronous signal and can be applied at any time, without regard to the clock waveform and also that this signal is level sensitive. As long as the clear input to the chip is held low, the flip-flop outputs will all remain low. (The register will contain all zeros!) Shifting data into the register in a serial fashion is exactly the same as the previously discussed 74LS91. Data at the serial inputs may be changed while the clock is either low or high, but the usual setup and hold times must be observed. The data sheet for this device gives setup time as 30 ns minimum and hold time as Data outputs C L 0 C K 54/74164 B ~ Data inputs QA '------~---..J Data outputs (a) DIP pinout CLEAR CLOCK A B Parallel data outputs (b) Logic diagram 54/7 4164 8-bit shift register Registers 0.0 ns. Since data are shifted into the register on PTs, the data input line must be stable from 30 ns before the PT until the clock transition is complete. Let's take a look at the gated serial inputs A and B. Suppose that the serial data is connected to A; then B can be used as a control line. Here's how it works: B is Held High The NAND gate is enabled and the serial input data passes through the NAND gate inverted. The input data is shifted serially into the register. B is Held Low The NAND-gate output is forced high, the input data stream is inhibited, and the next positive clock transition will shift a O into the first flip-flop. Each succeeding positive clock transition will shift another O into the register. After eight clock pulses, the register will be full of zeros! How long will it take to shift an 8-bit number into a 54164 shift register if the clock is set at IO MHz? Solution A minimum of eight clock periods will be required since the data is entered serially. One clock period is 100 ns, so it will require 800 ns minimum. For the register in Example 9.4, when must the input data be stable? When can it be changed? Solution The data must be stable from 30 ns before a positive clock transition untiHhe positive transition occurs. This leaves 70 ns during which the data may be changing(~~e Fig. 9.8). IOOns clock period · .· t PT --J Clock Q--,.--,-__. I . ·. · 70-ns . I_30-ns_ I r::::= data =:i-setu~ transition time time · The waveforms shown in Fig. 9.9 show the typical response of a 54/74164. The serial data is input at A (pin 1), while a gating control signal is applied at B (pin 2). The first clear pulse occurs at time A and simply resets all flip-flops to 0. The clock begins at time B, but the first PT does nothing since the control line is low. At time C the control line goes high, and the first data bit (a 0) is shifted into the register at time D. The next 7 data bits are shifted in, in order, at times E, F, G, H, I, J and K. The clock remains high after time K, and the 8-bit number 0010 1100 now resides in the register and is available on the eight output lines. This assumes that the LSB was shifted in first and appears at QH- Notice that the clock must be stopped after its positive transition at time K, otherwise shifting will continue and the data bits will be lost. Finally, another clear pulse occurs at time L, the flip-flops are all reset to zero, and another shift sequence may begin. Incidentally, the register can be cleared by holding the control line at B low and allowing the clock to run for eight PTs. This simply shifts eight Os into the register. Digital Pdnciples and Applications AB CD EFG HI J K L Titne~~1~ - , - ~ - . - . , - - - , . ~ - r - ~ - - . - ~ . - - - - , ~ - , - ~ . - ~ ~ ~ ~ - , -1 ~ Clear -1_J-+--,..-;---,.---;.--;--..;i--;--..;--;.-----u-' II Control (B) I L.I.STl- - ' - - - ! - - - - - - , . . . - Data (A) I I ----- Clock , I !...o....._______. 0 QA -=--.... I QB I :o =-._;0____________. I I ~ ----------- Qc --: O - ,.., I I QD --: 0 - : 0 ;-," - - - - - - - - - - - - - " " " " I =-: 0 ..----.:-l-----;~ Qp=-...... !0__________11 L£_ QE I I I ~,"--------------~ I I I QG ==: 0 I I I :0 : 0 I I QH =: =::;...o________________"""':_o_____...;:_o....,.. In prior sections, the ideas necessary for shifting data into and out of a register in serial have been developed. We can now use these same ideas to develop methods for the parallel entry of data into a register. There are a number of different techniques for the parallel entry of data, but we shall concentrate our efforts on commercially available TTL At first glance, the logic diagrams for some of the shift registers seem rather formidable (see, for instance, the block diagram for the 54/74166); but they aren't really. The 54/74166, for instance, is an 8-bit shift register, and the same circuit is repeated eight times. So, it's necessary to study only one of the eight circuits, and that's what we'll do here. The pinout and logic block diagram for a 54/74166 are given in Fig. 9.10. The functional description given on the TTL data sheet says that this is an 8-bit shift register, capable of either serial or parallel data entry, and serial data output. Notice that there are eight RS flip-flops, each with some attached logic circuitry. Let's analyze one of these circuits by starting with the RS flip-flops and then adding logic blocks to accomplish oμrneeds. Registers Clear Serial input - - ~ Shift/load A (2) Parallel Parallel inputs Shift/ Input O u t p u t ~ load H QH G F E Clear 15 Shift/ load H QH G F E Serial input CK Serial input A B ~----v---Parallel inputs Clock Clock GND inhibit Positive logic: see description (a) Pinout (14) Clock H-+(7,..)--1:---.. Clock inhibit (6) >:>-------+-' (b) Logic diagram 54/74166 Digital Principles and Applications First recognize that the clocked RS flip-flop and the attached inverter given in Fig. 9.1 la fonn a type D flip-flop. If a data bit Xis to be clocked into the flip-flop, the complement of X must be present at the input. For instance, if X = 0, then R = 0 and S = 1, and a 1 will be clocked into the flip-flop when the clock transitions. s Q (X) s ~--+--tR Q (X) ~--+--tR Clock---~ Clock---~ (a) Type D flip-flop (b) NOR-gate added .----.uo s Q Control Clock----' (c) Control logic added Q (1) Control= 1 (d) Example 9.5 Now, add a NOR gate as shown in Fig. 9.11 b. If one leg of this NOR gate is at ground level, a data bit X at the other leg is simply inverted by the NOR gate. For instance, if X = 1, then at the output of the NOR gate X = 0, allowing a 1 to be clocked into the flip-flop. This NOR gate offers the option of entering data from two different sources, either X 1 or X 2. Holding X2 at ground will allow the data at X 1 to be shifted into the flip-flop; conversely, holding X 1 at ground will allow data at X2 to be shifted in. The addition of two AND gates and two inverters as shown in Fig. 9. llc will allow the selection of data X 1 or dataX2. If the control line is high, the upper AND gate is enabled and the lower AND gate is disabled. Thus X 1 will appear at the upper leg of the NOR gate while the lower leg of the NOR gate will be at ground level. On the other hand, if the control line is low, the upper AND gate is disabled while the lower AND gate is enabled. This allows X 2 to appear at the lower leg of the NOR gate while the upper leg of the NOR gate is at ground level. You should now study this circuit until your understanding is crystal clear! Consider writing 0 or l at each gate leg in response to various inputs. To summarize: CONTROL is High CONTROL is Low Data bit atX1 will be shifted into the flip-flop at the next clock transition. Data bit atX2 will be shifted into the flip flop at the next clock transition. 0 Registers For the circuit in Fig. 9.11 c, write the logic levels present on each gate leg if CONTROL= 1, X1 = 1, andX2 = I. The cone(.lt1evels are given in parentheses in Fig, Sl.l l d, The data value J atX1is ,,hift,,,lin1to tl,,, nro~11on when the clock transitions. Solution A careful examination will reveal that exactly eight of the circuits given in Fig. 9 .11 c are connected together to form the 54/74166 shift register shown in Fig. 9.10. The only question is: how are they connected? The answer is: they are connected to allow two different operations: (1) the parallel entry of data and (2) the operation of shifting data serially through the register from the first flip-flop QA toward the last flip-flop QH, If the data input labeled X 2 in Fig. 9. llc is brought out individually for each flip-flop, these eight inputs will serve as the parallel data entry inputs for an 8-bit number ABCD EFGH. These eight inputs are labeled A, B, C, D, E, F, G, and Hin Fig. 9.10. The control line is labeled shift/load. Holding this shift/load control line low will enable the lower AND gate for each flip-flop, and the 8-bit number will be LOADED into the flip-flops with a single clock transition-PARALLEL input. Holding the shift/load control line high will enable the upper AND gate for each flip-flop. If the input from this upper AND gate receives its data from the prior flip-flop in the register, each clock transition will shift · a data bit from one flip-flop into the following flip-flop-proceeding in a direction from QA toward QH. In other words, data will be shifted through the register serially! In the first flip-flop in the register, the upper AND-gate input is labeled serial input. Thus data can also be entered into this register in a serial fashion. To summarize: Shift/Load is Low A single clock transition loads 8 bits of data (ABCD EFGH) into the register in parallel. Shift/Load is High Clock transitions will shift data through the register serially, with entering data applied at the SERIAL INPUT. Notice that the clock is applied Internal Levels Outputs Inputs through a two-input NOR gate. When Parallel Shift/ Clock QA and QB QH Clear Clock Serial clock inhibit is held low, the clock A ... H load inhibit signal passes through the NOR gate L X X X X X L L L inverted. Since the register flip-flops respond to NTs, data will shift into X L L X X H QBO QHO QAO the register on the PTs of the clock. t h a ... h b X a L L H When clock inhibit is high, the NOR- - - - - - - - ,-- - - - t H X H H L H gate output is held low, and the clock QA,, QGn is prevented from reaching the flipt L L X H L H QAn QGn flops. In this mode, the register can X H t H X X QHO QAO QBO be made to stop and hold its contents. A low level at the clear input can X = Irrelevant, H = High level, L = Low level t = Positive transition be applied at any time without regard a ... h = Steady state input level at A . . .H respectively to the clock, and it will immediately QAO• QBO = Level at QA, QB ..• before steady state reset all flip-flops to 0. When not in QA,,, QG,, Level of QA or QB before most recent transition ( ) t use, it should always be held high. The truth table in Fig. 9.12 sum54/7 4166 truth table maiizes the operation of the 54/74166 -- Digital Principles and Applications 8-bit shift register. You should study this table in conjunction with the logic diagram to understand clearly how the register can be used. Which entry in the truth table in Fig. 9.12 accounts for the parallel entry of data? 9.5 PARALLEL IN-PARALLEL out D1 (3) CK The fourth type of register discussed in the introductory section of this chapter is designed such that data can be shifted either into or out of the register in parallel. In fact, simply adding an output line from each flip-flop in the 54/74166 discussed in the previous section would meet the parallel in-parallel out requirements. [It would, of course, require a larger dual in-line package (DIP)-say, a 24-pin package.] Clear D (4) 2 CK Clear D (6) 3 CK The 54/74174 The 74174 in Fig. 9.13 is an example of a parallel in-parallel out register. The Texas Instruments data sheet refers to it as a hex D-type flip-flop with clear. It is simply a parallel arrangement of six D-type flipflops. Each flip-flop is negative-edge-triggered, and thus a PT will shift data into the register. The six data bits, D 1 through D 6 are all shifted into the register in parallel. The stored data is immediately available, in parallel, at the outputs, Q 1 through Q6. This type of register is simply used to store data, and is sometimes called a data register, or data latch. Notice that it is not possible to shift stored data either to the right or to the left. A low level at the clear input will immediately reset all flip-flops low. The clear input is asynchronous-that is, it can be done at any time and it takes precedence over all other inputs. The 74LS174 data sheet gives a setup time of 20 ns and a hold Clear D (11) 4 CK Clear D (13) 5 CK Clear iJ c14> 6 CK 54/74174 Registers time of 5 ns. What is the minimum required width of the data input levels (D1 ... D6) for the 74LS174 in Fig. 9.13? ?S Solution The data inputs m11.i,t be S:te~dy at least 20 before the PT of the clock, and they must be held for a minimum of 5 ns after the PT. Thus,.the da1;a inputlevels must be held steady fora minimum of25 ns (see Fig. 8.24 for comparison). The 54/74198 The 54/74198 is an 8-bit TTL MSI having both parallel input and parallel output capability. The DIP pinout for this device is given in Fig. 9.14 on the next page. It uses positive edge-triggered flip-flops, as indicated by the small triangle at pin 11. Notice that a 24-pin package is required since 16 pins are needed just for the input and output data lines. Not only does this chip satisfy the parallel input-output requirements; it can also be used to shift data through the register in either direction-referred to as shift right and shift left. All the registers previously discussed have the ability to shift right, that is, to shift data serially from the data input flip-flop toward the right, or from a flip-flop QA toward flip-flop Qs. We now need to consider how to shift left. Shift left serial Input input H Vee QH Input G QG Input F QF QH G QG F QF QE Clear 24 S1 L H E 54/74198 So QE Clear CK So Shift Input right A serial input Input B QB Input C Qe Input D QD 12 Clock GND 54/74198, 8-bit shift register. Parallel input-parallel output There are a number of 4-bit parallel in-,-parallel out shift registers available since they can be conveniently packaged in a 16-pin DIP. An 8-bit register can be created by either connecting two 4-bit registers in series or by manufacturing the two 4-bit registers on a single chip and placing the chip in a 24-pin package (such as the 54/74198). Let's analyze a typical 4-bit register, say, a 5417495A. The data sheet for the 5417495A describes it as a 4-bit parallel-access shift register. It also has serial data input and can be used to shift data to the right (from QA toward Qs) and in the opposite direction-,to the left. The DIP pinout and logic diagram are given in Fig. 9.15. The basic flip-flop and control logic used here are exactly the same as used in the 54/74164 as shown in Fig. 9.llc. The parallel data outputs are simply the Q sides of each of the four flip-flops in the register. In fact, note that the output QD could be used as a serial output when data is shifted from left to right through the register (right shift). Digital Principles and Applications Outputs Clock 2 ~--~'~--~Clock I L shift Vee QA QB Qe QD R shift (load) 14 CKI CK2 Serial input A B D C Mode 1 Serial A B D Mode GND input ~ - - - - . , . - - - ~ control Inputs (a) Pinout Data inputs A (2)(14) Clock I right shift Clock2 left shift B (3)(2) (4)(3) (5)(5) (8)(8) Note: The pin numbers in parentheses correspond to the ('95A, 'LS95) ('L95), respectively. (12)(12) QB Outputs (b) Logic diagram 54/7495A When the mode control line is held high, the AND gate on the right input to each NOR gate is enabled while the left AND gate is disabled. The data at inputs, A, B, C and D will then be loaded into the register on a negative transition of the clock-this is parallel data input. When the mode control line is low, the AND gate on the right input to each NOR gate is disabled while the left AND gate is enabled. The data input toflip-flop QA is now at serial input; the data input to Qs is QA and so on down the line. On each clock NT, a data bit is entered serially into the register at the first flip-flop QA, and each stored data bit is shifted one flip-flop to the right (toward the last flip-flop Qv). This is the serial input of data (at serial input), and also the right-shift operation. Registers In order to effect a shift-left operation, the input data must be connected to the D data input as shown in Fig. 9.16 below. It is also necessary to connect QD to C, Qc to B, and QB to A as shown in Fig. 9.16. Now, when the mode control line is held high, data bit will be entered into flip-flop QD, and each stored data bit will be shifted one flip-flop to the left on each clock NT. This is also serial input of data (but at input D) and is the left-shift operation. Notice that the connections described here can either be hard wired or can be made by means of logic gates. Serial data input (95, 'LS95) ('L95) A (2)(14) Data inputs B (3) (2) C (4)(3) D (5)(5) Clock I right shift Clock 2 left shift (12)(12) QB (11)(10) Outputs Qc (10)(9) QD 54/7 495A wired for shift left There are two clock inputs--clock 1 and clock 2. This is to accommodate requirements where the clock used to shift data to the right is separate from the clock used to shift data to the left. If such a requirement is unnecessary, simply connect clock 1 and clock 2 together. The clock signal will then pass through the ANDOR gate combination noninverted, and the flip-flops will respond to clock NTs. Draw the waveforms you would expect if the 4-bit binary number l Ol O were shifted into a 5417495A in parallel. Solution The mode control line must be high, The data. input line~ must be stable for more than 10 ns prior to the clockNTs (setup timefor the data sheet information). A single clock NT will enter the data. (The waveforms are given in Fig. 9 .17.) If the clock is stopped after the transition time T, the levels on the input data lines may be changed, However, if the clock is not stopped, the input data line levels must be maintained. At this point, it simply cannot be overemphasized that the input control lines to any shift register must be controlled at all times! Remember, the register will do something every time there is a clock transition. What it does is entirely dependent on the levels applied at the control inputs. If you do not account for input control levels, you simply cannot account for the behavior of the register! Digital Principles and Applications ~ Mode Clock i_._ Jn L _ _ _ 0 ! A I 0 B 1 0----'------- c Outputs 1 0 Mode selection QD Clock S 1 S0 12 D 1 11 10 9 7 8 o---~,-----,' 1-------+:~~~~- Q 0-------~ A I I I-----------t-+-----,D >----+--+----iD Q (15) O -A CK Clear Serial input (2) for right shift --------------+rrri.-_./, CK Data C -'--'------11-1-+-+-------,1-~-r--..,,S l ~lear 'f DataD ~~1:>i (6) _LJ":. (1) Clear (10) l J QD >CK K QD Clear 'f - ~ (7) ENABLE; J - I)T (15) Ripple >---carry _J (a) 54/74160 (continued on next page) (!i) ______________ D_ig_i_ta_lP_n_·n_c_ip_le_s_a_nd_A_p_p_li_ca_t1_·on_s_ _ _ _ _ _ _ _ _ _ _ _ _ SN54160, SN54162, SN74160, SN74162 Synchronous binary counters Typical clear, preset, count, and inhibit sequences Illustrated below is the following sequence. 1. Clear outputs to zero. 2. Preset to BCD seven. 3. Count to eight, nine zero, one, two, and three. 4. Inhibit. μ (Asynchronous) Clear (SN54160, SN74160) Clear (SN54162, SN74162) -u~(-Sy_n_c_hr_o_n_o-us-·)--------------Load A _J B [=== [=== [-=== -- _J Data inputs C _J D -----'---~--Clock (SN54160, SN74I60) Clock (SN54162, SN74I62) ENABLE? Outputs I QD Carry =======~-;_ _,____, 17 I 18 9 0 1 2 31 Count - - - + . - - - Inhibit - Clear Preset (b) (Continued) Synchronous Up-Down Counters The 54/74193 is a 4-bit synchronous up-down binary counter. It has a master reset input and can be reset to any desired count with the parallel load inputs. The logic symbol for this TTL MSI is shown in Fig. I 0.29a. Pin PL is a control input for loading data into pins PA, Ps, Pc, and PD. When the device is used as a counter, these four pins are left open and PL must be held high. Pin MR is the master reset, and it is nonnally held low. (A high level on MR will reset all flip-flops.) Counters Outputs TCu and TCD are to be used to drive the following units, such as in a cascade arrangement. The clock inputs are CPu and CPD. Placing the clock on CPu will cause the counter to count up, and placing the clock on CPD will cause the counter to count down. Notice that the clock should be connected to either CPu or CPD, but not both, and the unused input should be held high. The outputs of the counter are QA, Q 8 , Qc and QD· A state diagram is a simple drawing which shows the stable states of the counter, as well as how the counter progresses from one count to the next. The state diagram for the 54/74193 is shown in Fig. 10.29b. Each box represents a stable state, and the arrows indicate the count sequence for both count-up and countdown operations. This is a 4-bit counter, and clearly there are 16 stable states, numbered 0, 1, 2, ... , 15. PL CPu P.4 Ps Pc PD TCu 54/74193 CPD MR TCD QA QB Qc QD (a) --Count up - - - - Count down (b) 4-bit binary counter (presettable) The 54/74193 has a parallel-data-entry capability which permits the counter to be preset to the number present on the parallel-data-entry inputs (P,1, P 8 , Pc, and PD). Whenever the parallel load input (PL) is low, the data present at these four inputs is shifted into the counter; that is, the counter is preset to the number held by PDPcPsPA. Now, here is another technique for modifying the count. Simply use a NAND gate to detect any of the stable states, say, state 15 (1111), and use this gate output to take p L low. The only time p L will be low is when QD, Qc, Q8 , and Q,1 are all high, or state 15(1111). At this time, the counter will be preset to the data PvPcPsPA. For example, suppose that PvPcPsPA = 1001 (the number 9). When the clock is applied, the counter will progress naturally to count 15(1111). At this time, PL will go low and the number 9 (1001) will be shifted into the counter. The counter will then progress through states 9, 10, 11, 12, 13, and 14, and at count 15 it will again be preset to 9. The count sequence is easily shown by the state diagram in Fig. 10.30 on the next page. Notice that count 15 (1111) is no longer a stable state; it is the short time during which the counter is preset. The stable states in this example are 9, 10, 11, 12, 13, and 14. This is, then, a mod-6 counter. Notice that this technique is Digital Principles and Applications asynchronous since the preset action is not in synchronism with the clock. Therefore, you should be aware that counting spikes or glitches may be associated with the outputs of this presetting arrangement. Suppose that the counter just discussed is still preset to I 001 (the number 9) but the clock is applied to count down rather than count up. What are the counting states? What is the modulus? Solution The counter will count down to 15, then preset back to 9, and repeat. The resulting state diagram is given in Fig. l 0.31. The modulus is clearly 10. 0 (is) ~ 2 3 4 5 6 7 8 12. 13. 14. 15. Name two popular synchronous binary counters. What is the difference between the 74161 binary counter and the74191binary counter? What is the modulus of the 74160 counter? Can a 74160 counter be used to count down? 10.7 COUNTER DESIGN AS A SYNTHESIS PROBLEM Section 8.11 of Chapter 8 presents a systematic approach towards sequential logic circuit design using FSM concept. In this section, we consider counter as a state machine and discuss counter design steps through an example. Let us try to design a modulo-6 counter, the counting states (memory values) of which are shown in state transition diagram of Fig. l 0.32. We need three memory elements or flipflops for this as with n flip-flop we can get at most 2" number of different counting states. Now with three flip-flop, 8 different states are possible but in our design states 110 and 111 are not used in the counting State sequence of a modulo-6 counter Counters sequence. To start with we shall assume the counter is always initialized with one of the valid states and not 110 or 111. We decide to use three JK flip-flops labeled A, B and C as memory element for this design. The next step to be taken is to form a state synthesis table as shown in Table 10.1. In this, the first column represents current state of the counter and second column, as shown in the next state of the counter state transition diagram. We fill up next three columns using excitation table of JK flip-flop given in Fig. 8.34 of Chapter 8. Excitation table gives inputs need to be present when clock triggers a certain Qn~Q,,+ 1 transition of the flip-flop. In the first row, we see both C and B make transition O~O and hence corresponding JK inputs should be Ox from excitation table. For flip-flop A, transition is 0~ I and input should be Ix. This is continued to fill up other five rows of input columns for three flip-flops. State Table for Design of Modulo-6 Counter Given in Fig. 10.32 c,, B,, A,, Cn+J Bn+l An+l Jc Kc ls Ks 0 0 0 0 I I 0 0 1 I 0 0 0 I 0 I 0 1 0 0 0 1 I 0 0 I 1 0 0 0 I 0 1 0 1 I 0 0 0 X X 0 1 X X I X X 0 I X JA K,1 X 1 X X X I 0 1 X X 1 X 1 0 0 X I X X X l Our next objective is to get logic equation for each flip-flop input as a function of present state of the counter. We use Kamaugh Map for this as shown in Fig. 10.33. Note that values corresponding to unused states 110 and 111 appear as don't care 'x'. We have not shown Kamaugh Map for JA and K._4 as it is obvious from Table 10.1 thatJA =KA= I. e" BnAn 00 0I 0 0 0 I X X 11 - I0 e11 Bn-A" 00 0I l 1 10 1 0 0 X X X X X X I 0 I X X - Kc=A 11 e/l BA n n 00 01 I1 I0 xi X X X 0 0 I1 1 0 0 e" BIIA,, 00 0 0I X X X X I I 10 0 X X Ks=A,, Derivation of design equations from Karnaugh Map The final step is to draw the circuit diagram from these design equations, which is shown in Fig. 10.34. The decoding output is obtained from a three input AND gate which goes high every time the counter goes to a valid state CBA = 000 and that occurs in every 6th clock cycle. Note that the method we have explained is a general one and can be used to design counter of any modulo number and that can follow any given counting sequence. An irregular counter is the one which (;) _ _ _ _ _ _ _ _ _ _ _ _ _ _D_ig_it_al_P_n_·nc_ip_le_s_a_n_d_A_pp_lJ_·ca_t_io_ns_ _ _ _ _ _ _ _ _ _ _ _ _ does not follow any regular binary sequence but has N number of distinct states and thus qualifies as a modulo-N counter. In Example 10.15, we present a modulo-4 irregular counter. One question can be raised at this point for the above circuit. What happens if the circuit for any reason goes to one of the unused state? Does it come back to any of the valid counting state or in the worst case gets locked as shown in Fig. 10.35a? Initializing the designed circuit with 110 or 111 unused state we find that they get back to counting sequence as shown in Fig.10.35b. However, a designer may not leave unused states to chances and want them to follow certain course if the circuit accidentally enters into one of them. Example 10.14 shows how to handle unused states in a counter design problem. JA K.1 A A B A B Jc C c CLK y Circuit diagram of modulo-6 synchronous counter described in Fig. 10.32 (a) (b) (a) Lock-in conditions, (b) Full state transition diagram for circuit in Fig. 10.34 Desig,1 a self-correcting rnodulo-6 counter as described in Fig. 10.32 in which all the unused state leads to state CEA = 000. For this we have to add two more rows as given next for two unused states Counters Accordingly, Kamaugh Map giving design equations changes to as given in Fig. 10.36. C11 BA 11 11 BA 11 11 00 01 11 10 C/l BA 11 Tl 00 ~ 0 0 0 1 X X 1 0 0 X X X 1 0 ~ 01 Ix I1 11 xi 1I 10 CI! 00 01 11 10 xi X X X X 0 0 I1 1 1 0 0 Kc=A 11 + B,, c BA 11 B,,A,, 11 0 Js=C11An BA I! 11 /l oo 01 X X X X 11 10 0 CII 00 01 l 1 l0 0 1 X X 1 1 l X X 0 I C/l 00 01 11 10 0 X 1 1 X 1 X 1 l X Kn=A 11 +C11 Design equations for Example 10.14 Note the difference between Fig. 10.33 and 10.36. Unused states 110 and 111 can no longer be considered as don't care. This type of design is called self-correcting as the circuit comes out on its own from an invalid state to a valid counting state sequence. The final circuit diagram from design equations are shown in Fig.10.37. C c y Circuit diagram for Example 10.14 Design a modulo-4 irregular counter with following counting sequence using D flip-flop. 00-10-11-01 Soliitidn 10.2. Using state excitation table of D flip-flop (Fig. 8.34), the state table can be formed as shown in Table Digital Principles and Applications State Table for Design of Irregular Counter Bn An B,,+1 A,,+1 Ds D.4 0 0 1 l 0 l 0 1 0 l 0 0 0 l 0 0 0 I 0 Design equations from Kamaugh Map can be derived as shown in Fig. 10.38(a), and con-esponding logic circuit is shown in Fig. I0.38(b). A B,, An B,, 0 0 0 0 Flip-flop A 0 CLK 0 Ds=A,, D;1=Bn ~---iD Q An 0 0 D B Q Flip-flop B Q CLK Q Clock (a) (b) (a) Deriving design equations for Example 10.15, (b) Circuit diagram Show how a modulo-4 counter designed with two flip-flops can generate a repetitive sequence of binary word '1101' with minimum number of memory elements? Solution Let the counting sequence oftwo flip-flops B and A be 00 -+ 0 I -+ I O ~ 11.-+ 00 ... , Le. a modulo-4 synchronous up counter. The corresponding output is l-+ 1-+ 0-+..l-+ L .. As shown in Fig. 10.39(a) the sequence '1101' will be generated.repetitively by Y. Figure 10.39(b) gives Kamaugh Map representation of Y and we get Y = A + B'. A standard modulo-4 up counter and an 2sinput OR gate connected as shown in Fig. 10.39(c) generates the given sequence. Note that for N-bit sequence generator we need modulo-N counter. Modulo-N synchronous counter requires m number of flip-flops where mis the lowest integer for which 2 N. The design procedure remains the same as discussed in Example 10.16. Output Y now is a function of m state variables representing m memory elements. Compare this design with shift register based sequence generator design discussed in Chapter 9 ·that requires N number of memory elements for N-bit sequence generator. Though shift register based design does not require any combinatorial circuit to generate output logic the overall hardware cost is more and it is more pronounced for large N. A similar design for sequence detector circuit with minimum number of flip-flops is discussed in Chapter 11. 111 ::::: 16. What is lock-out of a counter? 17. For 48-bit sequence generator what is the minimum number of memory elements required? Counters A 0 0 l 1 B 0 1 0 l AWO y 1 I 0 1 I 0 1 0 1 1 1 Y=A+B (b) (a) ------10 ll 10 ll '--v---' '--v---' J A i--~~-1_. J y CLK-,----0!> K K A (c) Sequence generator circuit using synchronous counter, (a) State Table, (b) Output equation, (c) Circuit diagram 10.8 A DIGITAl CLOCK A very interesting application of counters and decoding arises in the design of a digital clock. Suppose that we want to construct an ordinary clock which will display hours, minutes, and seconds. The power supply for this system is the usual 60-Hz 120-Vac commercial power. Since the 60-Hz frequency of most power systems is very closely controlled, it is possible to use this signal as the basic clock frequency for our system. Note that in several countries commercial power supply is 50-Hz and not 60-Hz. There one can use standard variable frequency signal generator, set at 60-Hz, as input. In order to obtain pulses occurring at a rate of one each second, it is necessary to divide the 60-Hz power source by 60. lfthe resulting 1-Hz waveform is again divided by 60, a one-per-minute waveform is the result. Dividing this signal by 60 then provides a one-per-hour waveform. This, then, is the basic idea to be used in forming a digital clock. A block diagram showing the functions to be performed is given in Fig. 10.40. The first divide-by-60 counter simply divides the 60-Hz power signal down to a 1-Hz square wave. The second divide-by-60 counter changes state once each second and has 60 discrete states. It can, therefore, be decoded to provide signals to display seconds. This counter is then referred to as the seconds counter. 60Hz -c:P'- 1 cycle/s __n_n_ +60 +60 Seconds counter ~r+60 ~k_cy_c_le_/h-r Minutes counter Block diagram of digital clock .ff~ Hours counter Digital Principles and Applications The third divide-by-60 counter changes state once each minute and has 60 discrete states. It can thus be decoded to provide the necessary signals to display minutes. This counter is then the minutes counter. The last counter changes state once each 60 minutes (once each hour). Thus, if it is a divide-by-12 counter, it will have 12 states that can be decoded to provide signals to display the correct hour. This, then, is the hours counter. As you know, there are a number of ways to implement a counter. What is desired here is to design the counters in such a way as to minimize the hardware required. The first counter must divide by 60, and it need not be decoded. Therefore, it should be constructed in the easiest manner with the minimum number of flipflops. For instance, the divide-by-60 counter could be implemented by cascading counters (12 x 5 = 60, or 10 x 6 = 60, etc.). The TTL MSI 7490 decade counter can be used as a divide-by-IO counter, and the TTL MSI 7492 can be used as a divide-by-6 counter. Cascading these two will provide a divide-by-60 counter as shown in Fig. 10.41. The amplifier at the input provides a 60-Hz square wave of the proper amplitude to drive the 7490. The 7492 is connected as a divide-by-12 counter, but only outputs QA, Q8 , and Qc are used. In this fashion, the 7492 operates essentially as a divide-by-6 counter. ~ / / Amplifier 60Hz 6Hz 1 Hz ...IISL ...IISL ...IISL (Qc) [/ Divide-by-60 counter The seconds counter in the system also divides by 60 and could be implemented in the same way. However, the seconds counter must be decoded. We are interested in decoding this counter to represent each of the 60 s in 1 min. This can most easily be accomplished by constructing a mod~ IO counter in series with a mod-6 counter for the divide-by-60 counter. The mod- IO counter can then be decoded to represent the units digit of seconds, and the .mod-6 counter can be decoded to represent the tens digits of seconds. Since both the 7490 and the 7492 count in straight 8421 binary, a 7447 decoder-driver can be used with each to drive two 7-segment indicators, as shown in Fig. I 0.42. Notice that the 7492 is connected as a divideby-12 counter; but only outputs QA, Q8 , and Qc are used to drive the 7447 decoder-driver. The minutes counter is exactly the same as the seconds counter, except that it is driven by the one-perminute square wave from the output of the seconds counter, and its output is a one-per-hour square wave, as shown in Fig. 10.42. The divide-by-12 hours counter must be decoded into 12 states to display hours. This can be accomplished by connecting a mod-IO (54/74160) decade counter in series with a single flip-flop E as shown in Fig. 10.43. This forms a divide-by-20 (10 x 2 = 20) counter. Feedback is then used to form a mod-12 counter. The hours counter must count through states 00, 01, 02, ... , 11, and then back to 00. The NAND gate in Fig. 10.43 will go low as the counter progresses from count 11 to count 12, and this will immediately clear the 74160 to 0000 and reset the flip-flop E to 0. The counter actually skips from count 11 to count 00 omitting the eight counts in between. This is the mod-12 hours counter; the 74160 will provide the units of hours while the flip-flop will provide the tens of hours. Notice that the 74160 is reset asynchronously and there might then be glitches at the outputs of the decoding gates. However, this is one case where these glitches will have no effect, since they are too narrow to cause a visible indication on the light emitting diodes (LEDs). Counters (1 Hz) (1 cycle/min) (1 cycle/h) _n__n_ +6 7492 7490 QD QC QB QA QD QC QB QA 7447 7447 ,-1 /_/ LED /-/ I I Tens Units + 10 (1 cycle/min) LED A 1O x 6 mod-60 counter with units and tens decoding RESET I cycle/h 10 74160 + CLR __n__n_ QD QC QB QA 7447 7447 Tens Units Mod-12 hours counter Finally, some means must be found to set the clock because the flip-flops will assume random states when the power is turned off and then turned back on again. Setting the clock can be quite easily accomplished by means of the SET push-buttons shown in Fig. 10.44. Depressing the SET HOURS button causes the hours counter to advance at a one-count-per-second rate, and thus this counter can be set to the desired hour. The minutes counter can be similarly set by depression of the SET MINUTES button. Digital Principles and Applications Hours + 12 Set Minutes minutes Set hours +60 -...._,_,.,e:---, 10 X 6 Set Seconds seconds 60 10 X 6 + --:S.Z...J.J,lt---, lHz +60 60Hz ,-, lol /_/ ~ Tens Units Tens Units Digital clock Depression of the SET SECONDS button removes the signal from the seconds counter, and the clock can thus be brought into synchronization. By means of large-scale integration (LSI), it is possible to construct a digital clock entirely on one semiconductor chip. Such units are commercially available, and they perfom1 essentially the function shown in the logic diagram in Fig. 10.43 (the seven-segment indicators are, of course, separate). The National Semiconductor 5318 is one such commercially available LSI digital clock. It is available in a 24-pin dual inline (DIP) package measuring 0.54 x 1.25 in. 10.9 COUNTER DESIGN USING HDl Counter design in HDL is straight forward if one uses arithmetic operator+ and - that corresponds to binary addition and subtraction respectively. We show a modulo-8 up counter design in the example given in first column. It is left to the compiler to decide which flip-flop is to be used. If one wants to ensure use of a particular type of flip-flop say, JK then the code should be written in a manner shown in second column for modulo-3 up counter shown in Fig. 10.16a. module UC(Clock, Reset,Q); input Clock, Reset; output [2:0] Q; //modulo 8 requires 3 flip-flop reg [2 : 0 J Q; always@ (negedge Clock or negedge Reset) if Q"'3'b0; else Q= Q+l; endmodule module UCJK(A,B,Clock,Reset); requires 2 flip-flop wire JA, JB, KA, KB; as.sign assign KA=l'bl; Counters module JKFF(Q,J,K,Clock,Reset); input·· .J, K, Clock, Reset; output Q; reg Q; always@ (negedge or negedge if{,-Reset) Q=l'bO; else Q ----~ CLK Analog input Vx Comparator '---------ic--l Ramp * Ramp gen +10 100 s 1 Control RESET _[-:' =MANUAL RESET Strobe 7-seg. decoder 7-seg. decoder 7-seg. decoder (a) One f,< . C ••..... ·... · . 4, Does the code inl'ig. 13.(ib have even or odd parity? ·‥ Ma.gnetic tape provides inexpensive storage oflarge qU!lntities of digittl! ~ta.)Vhy rttuse i(instead ofRAM, in a microcon1putel'? 6: How can binary information be recordca! CMOS inverter is a plot of input voltage versus .output voltage (Fig. 14.38). 23. Its purpose is to raise the minimum TTL high output level above the lowest allowable CMOS high input kweL 24. Alevel shifter is used between a TTL gate driving a CMOS gate; it is used to make their high and low levels compatible. 25. 'Yes,no 26. The CMOS has current sink and source limitations. Applications + + + + + + Understand the multiplexing techniques used with LED displays List and describe the main sections of a frequency counter Explain how a time measurement circuit can be designed Be familiar with the basic features of the ADC0804 A/D converter Be familiar with the basic features of the ADC3511 microprocessor compatible AID converter Discuss how to construct a digital voltmeter using the National Semi-conductor ADD3501 chip This chapter is intended to tie together many of the fundamental ideas presented previously by considering some of the more common digital circuit design encountered in industry. The multiplexing of digital LED displays is considered first since it requires the use of a number of different TTL circuits studied in detail in prior chapters. Digital instruments that can be used to measure time and frequency are considered next, and the concept of display multiplexing is applied here. A number of applications using the popular ADC0804 are presented. An intergrating-type converter, the microprocessor-compatible ADC3511, is studied in detail. Then a similar converter, the ADC3501, is used to construct a digital voltmeter. In most of the applications considered, specific TTL part numbers have been specified, but in the interest of clarity, detailed designs including pin numbers have not been provided. However, it is a simple matter to consult the appropriate data sheets for this information. In some cases, a specific part number has not been assigned; an example of this is the 1-MHz clock: oscillator shown in Fig. 15.14, or a divide-by-IO counter in the same figure. In such cases, it is left to you _ _ _ _ _ _ _ _ _ _ _ _ _ _A_p_pl_ica_ti_·on_s _ _ _ _ _ _ _ _ _ _ _ _ _ _ ~ to select any one of a number of divide-by- IO circuits, or to choose an oscillator circuit such as discussed in a previous chapter, on the basis of availability, cost, ease of use, compatibility with the overall system, and other factors. 15.1 MULTIPLEXING DISPLAYS The decimal outputs of digital instruments such as digital voltmeters (DVMs) and frequency counters are often displayed using seven-segment

indicators. Such indicators are constructed by using a fluorescent bar, a liquid crystal bar, or a LED bar for each segment. LED-type indicators are convenient because they are directly compatible with TTL circuits, do not require the higher voltages used with fluorescents, and are generally brighter than liquid crystals. On the other hand, LEDs do generally require more power than either of the other two types, and multiplexing is a technique used to reduce indicator power requirements. The circuit in Fig. 15.la is a common-anode LED-type seven-segment indicator used to display a single decimal digit The 7447 BCD to seven-segment decoder is used to drive the indicator, and the four inputs to the 7447 are the four-flip-flop outputs of the 7490 decade counter. Remember that the 7447 has active low outputs, so the equivalent circuit of an illuminated segment appears as in Fig. 15.lb. A 1-Hz square wave applied at the clock input of the 7490 will cause the counter to count upward, advancing one count each second, and the equivalent decimal number will appear on the display. A similar single decimal digit display using a common-cathode-type LED indicator is shown in Fig. 15.2a. The seven-segment decoder used here is the 7448; its outputs are active high, and they are intended to drive buffer amplifiers since their output current capabilities are too small to drive LEDs directly. The seven npn transistors simply act as switches to connect +Vee to a segment. When an output of the 7448 is high, a transistor is on, and current is supplied to a LED segment. The equivalent circuit for an illuminated segment is shown in Fig. 15.2b. When an output of the 7448 is low, the transistor is off, and there is no segment current and thus no illumination. Let's take a look at the power required for the single-digit display in Fig. 15.la. A segment is illuminated whenever an output of the 7447 goes low (essentially to ground). Ifwe assume a 2-Vdc drop across an illuminated segment (LED), a current I= (5 - 2)/150 = 20 mA is required to illuminate each segment. The largest current is required when the number 8 is displayed, since this requires all segments to be illuminated. Under this condition, the indicator will require 7 x 20 = 140 rnA. The 7447 will also require about 64 mA, so a maximum of around 200 mA is required for this single digit display. An analysis of the display circuit in Fig. 15.2 will yield similar results. A digital instrument that has a four-digit decimal display will require four of the circuits in Fig. 15 .1 and thus has a current requirement of 4 x 200 = 800 mA. A six-digit instrument would require 1200 mA, or 1.2 A, just for the displays! Clearly these current requirements are much too large for small instruments, but they can be greatly reduced using multiplexing technique. Basically, multiple,dng is accomplished by applying current to each display digit in short, repeated pulses the pulse repetition rate is sufficiently high, your eye will perceive a steady rather than continuously. illumination without any flicker. (For instance, hardly any flicker is noticeable with indicators illuminated using 60 Hz.) Thesingle-digit display in Fig. 15.3a has +5 Vdc (and thus current) applied through a pnp transistorthat acts as a switch. When DIGIT is high, the transistor (switch) is off, mid the indicator current is zero. When DIGIT is low, the transistor is on, and a number is displayed. If the wavefonn in Fig. 15.3b is used as DIGIT, the transistor will be on and the segment will display a number for only 1 out of every 4 ms. Even though the display is not illuminated for 3 out of 4 ms, the illumination will appear to your eye as if it ff Digital Principles and Applications ClockCommon anode type LED display R(l50 Q, typical) 7490 Decade counter ABC D ABC D 7448 7-segment decoder abcdefg .... ~----1--_..-+Vee abcdefg 7447 7-segment decoder ABC D Clock ABC D 7490 Decade counter R (150 Q, typical) abcdefg Common anode type LED display (a) Single decimal-digit display (a) *1 +Vee ~LED segment R = (150 Q typical) R= 150.Q ~ (b) Equivalent circuit for an illuminated segment LED segment (b) Equivalent circuit for an illuminated segment were continuous. Since the display is illuminated with a pulse that occurs once every 4 ms, the repetition rate (RR) is given as RR= 1/0.004 = 250 Hz. As a guideline, any RR greater than around 50 or 60 Hz will provide steady illumination without any perceptible flicker. The great advantage here is that this single-digit display requires only one-fourth the current of a continuously illuminated display. This then is the great advantage of multiplexing! Let's see how to multiplex the four-digit display in Fig. 15.4a. Assume that the four BCD inputs to each digit are unchanging. If the four waveforms in Fig. 15.4b are used as the four DIGIT inputs, each digit will be illuminated for one-fourth of the time and extinguished for three-fourths of the time. Looking at the time line, we see that digit 1 is illuminated during time t" digit 2 during time t2, and so on. Clearly, ! 1 = t2 = t3 = 14 Applications +Vcc=+S Vdc DIGIT Common anode type LED l-4ms--l +Vc:=-u--u1 ms R (150 Q, typical) -I I- 3 ms -l (b) DIGIT waveform a b e d e 7447 ?-segment decoder A B C D (a) Multiplexed display = T/4. The repetition rate is given as RR= 1/T, and if the rate is sufficient, no flicker will appear. For instance, if t 1 = 1 ms, then T= 4 ms, and RR= 1/0.004 = 250 Hz. Now, here is an important concept; an illuminated digit requires 200 mA, and since only one digit is illuminated at a time, the current required from the +Vee supply is always 200 mA. Therefore, we are illuminating four indicators but using the current required of only a single indicator. In fact, in multiplexing displays in this way, the power supply current is simply the current required of a single display, no matter how many displays are being multiplexed! Explain the timing for a six-digit display that has a repetition rate of 125 Hz. Solution A'nRR of 125 Hz means that all digits must bg serviced once eyery 1~5 = 8 ms. Dividing the time equally among the six digits means that each digit will be on for 6 = 1.33 ms and off for 6.67 ms. Note that as the pulse width is decreased, the display• brightness will also decrease. It may thus become necessary to increase the peak current through each segment by reducing the size of the resistors R in Figs. IS.I and 15.2. The circuit in Fig. 15.3 show how to multiplex a common-anode-type display. Show how to multiplex a common-cathode-type display. The npn transistorin l'ig. 15 S is us~d as a switch betw~en the cathode of the display and ground. When the. transistor is on, current.is allowed to pa~s throu@a segment for illumination. When the transistor is off, no current is allowed, and the segm~nt cannot illuminate. The DIGIT waveform isshmvn in Fig. 15,Sb. Notice that positive pulse.is requireWhell t= the counterwill .display 7500 (transitions per second) x ·.· 1 (second) e= 7500. When t= IO s,the counterwill display 7 500 (transitions per second). x l 0. (seconds) = 75,000. For this last case; we would have to haveD-1~Hl /•1" ,,ht dit!it;i! m"1mnd,·r A low-cost DMM using the ADD 3501 (National Semiconductor Data Acquisition Handbook) ::i '" Applications The different de and ac voltage ranges are accommodated by a resistive voltage divider at the analog input. Alternating-current voltages are measured by using the three operational amplifiers A 3,A;" and A 5 to develop a de voltage that is proportional to the root-mean-square (RMS) value of the ac input voltage. Range Frequency Measurement Accuracy response mode 2.0 200 0.2 20 2000 DC volts V V V V :S: 1%FS AC volts VRMS VRMS VRMS VRMS 40 Hzto 5 kHz :S: 1%FS DC amps :S: 1%FS mA mA mA mA mA ACamos mARMS mARMS mARMS mARMS mARMS 40 Hz to 5 kHz :S: 1%FS kQ kQ kQ kQ Ohms kQ :S: 1% FS Overrange display ±OFLO +OFLO ±OFLO +OFLO +OFLO Performance of the DMM in Fig. 15.28 A series of current-sensing resistors are used to measure either de or ac current. The current to be measured is passed through one of the sensing resistors, and the DMM digitizes the voltage developed across the resistor. The DMM measures resistance by applying a known current from an internal current source (operational amplifiers A 1 and A) to the unknown resistance and then digitizing the resulting voltage developed. For those interested in pursuing this subject, complete details for the construction and calibration of this DMM are given in the National Semiconductor Data Acquisition Handbook. The pri~ary objective of fhischapter is to demonstrate the use of ~any of the most fundamental prindples discussed throughout the text by considering some of the more common digital circuit configurations encountered in industry. Multiplexing of LED displays, time and frequency measurement, and use of digital voltmeters ofall types are widely used throughout industry.Although our coverage is byno means comprehensive, it will ~rve as an excellent introduction to industrial prE7H E6H 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 11: 18 l9 20 00010111 00011000 00011001 00011010 . . OQOllQll lllO 0110 1110 0101 Hexadecimal B9H ESH Decimal -0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 -12 -13 -14 -15 -16 -17 -18 -19 -20 -21 -22 -23 -24 -25 -26 -27 Digital Principles and Applications . · .... / ..... . Positive Negative Decimal Hexadecimal Binary Bina!)' Hexadecimal Decimal 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 lCH !DH !EH lFH 20H 21H 22H 23H 24H 25H 26H 27H 28H 29H 2AH 2BH 2CH 2DH 2EH 2FH 30H 31H 32H 33H 34H 35H 36H 37H 38H 39H 3AH 3BH 3CH 3DH 3EH 3FH 40H 41H 42H 43H 44H 45H 0001 llOO 00011101 00011110 00011111 00100000 0010 0001 0010 0010 0010 0011 00100100 0010 0101 0010 0110 0010 0111 0010]000 00101001 00101010 00101011 0010 1100 0010 1101 00101110 00101111 00110000 0011 0001 00110010 001100ll OOll 0100 OOUOIOl 0011 OHO 00110111 00111000 OOll 1001 00111010 0011 1011 oou 1100 OOll llOl OOH 1110 0011 Illl 01000000 01000001 01000010 01000011 OlOO 0100 moo 0101 1110 0100 1110 0011 lllO 0010 1110 0001 1110 0000 11011111 1101 1110 1101 1101 1101 1100 1101 1011 1101 1010 1101 1001 1101 1000 11010111 1101 OllO 1101 0101 1101 0100 1101 0011 1101 0010 1101 0001 1101 0000 1100 l lll 1100 1110 1100 1101 llOO 1100 llOO lOll llOO 1010 1100 1001 1100 1000 1100 0111 1100 0110 llOO 0101 1100 0100 1100 0011 1100 0010 1100 0001 1100 0000 1011 llll lOll lllO 1011 1101 1011 1100 10111011 E4H E3H E2H ElH EOR DFH DEH DDH DCH DBH DAH D9H D8H D7H D6H D5H D4H D3H D2H DlH DOH CFH CEH CDH CCH .CBH CAH C9H C8H C7H C6H C5H C4H C3H C2H -28 -29 -30 -31 -32 -33 -34 -35 -36 -37 -38 -39 -40 -41 -42 -43 -44 -45 -46 -47 -48 -49 -50 -51 -52 -53 -54 -55 -56 -57 -58 -59 -60 -61 -62 -63 -64 -65 -66 -67 -68 -69 58 59 60 61 62 63 64 65 66 67 68 69 cm COH BFH BEH BDH BCH BBH _ _ _ _ _ _ _ _ _ _ _A_pp_e_nd_ix_2_:_2'_s_co_m_p_le_m_en_t_Re_p_re_se_n_ta_tio_n _ _ _ _ _ _ _ _ _ _ _ l Positive Negative Decimal Hexadecimal Bina1y Binary 70 71 46H 47H 48H 49H 4AH 4BH 4CH 4DH 4EH 4FH 50H 51H 52H 53H 54H 55H 56H 57H 58H 59H 5AH 5BH SCH 5DH 5EH 5FH 60H 61H 62H 63H 64H 65H 66H 67H 68H 69H 6AH 6BH 6CH 6DH 6EH 6FH 01000110 OlOOOHl 0100 1000 01001001 0100 1010 0100 1011 0100 HOO 0100.1101 0100 1110 0100 1111 0101 0000 01010001 0101 0010 0101 0011 01010100 01010101 0101 OHO 0101 om 0101 1000 01011001 0101 1010 Otol 10ll 0101 1100 0101 1101 0101 lllO 0101 1111 0110 0000 0110 0001 01100010 OllO 0011 0110 0100 0110 0101 OllO 0110 0110 0111 0110 1000 0110 1001 OHO 1010 0110 1011 0110 llOO 0110 1101 0110 1110 OllO ll 11 1011 1010 1011 1001 1011 1000 1011 0111 1011 OllO 1011 0101 1011 0100 1011 OOll 1011 0010 1011 0001 lOll 0000 1010 1111 1010 1110 1010 1101 1010 1100 1010 1011 1010 1010 1010 1001 1010 1000 1010 0111 1010 0110 1010 0101 1010 0100 1010 0011 1010 0010 1010 0001 1010 0000 1001 1111 1001 lllO 10011101 1001 1100 1001 1011 1001 1010 1001 1001 1001 1000 1001 0111 1001 0110 1001 0101 1001 0100 1001 0011 1001 0010 1001 0001 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 IOI 102 103 104 105 106 107 108 109 110 ll1 ~ Hexadecimal BAH B9H B8H B7H B6H B5H B4H B3H B2H BlH BOH AFH AEH ADH ACH ABH AAH A9H ASH A7H A6H ASH A4H A3H A2H AlH AOH 9FH 9EH 9DH 9CH 9BH 9AH 99H 98H 97H 96H 95H 94H 93H 92H 91H Decimal -70 -71 -72 -73 -74 -75 -76 -77 -78 -79 -80 -81 -82 -83 -84 -85 -86 -87 -88 -89 -90 -91 -92 -93 -94 --95 -96 -97 -98 -99 -100 101 -102 103 -104 105 -106 -107 -108 -109 -110 -111 Digital Principles and Applications Positive .· Decimal Hexadecimal Bina1y Binary 112 113 114 70H 7IH 72H 73H 74H 75H 76H 77H 78H 79H 7AH 7BH 7CH 7DH 7EH 7FH - 01110000 01110001 0111001() Olll 0011 om 0100 0111 0101 0111 0110 Olll Olll Olli 1000 01111001 Olll 1010 0111 lOll 0111 1100 Olll 1101 0111 1110 Ollllll 1001 0000 1000 1111 1000 1110 1000 1101 1000 1100 1000 1011 1000 1010 1000 1001 1000 1000 1000 0111 1000 0110 1000 0101 1000 0100 1000 0011 1000 0010 1000 0001 1000 0000 Il5 I I Negative 116 117 118 119 120 121 122 123 124 125 126 127 128 .- Hexadecimal 90H 8FH 8EH 8DH 8CH 8BH 8AH 89H 88H 87H 86H 85H 84H 83H 82H 81H 80H Decimal -112 -113 -114 -115 -116 -117 -118 -119 -120 -121 -122 -123 -124 -125 -126 -127 -128 Appendix 3: TTL Devices Number 7400 7401 7402 7403 •7404 •7405 7406 7407 7408 7409 7410 7411 ,7412 7413 7414 7416 7417 7420 7421 r1fZ2 7423. j742~ '7426 7427 7428 7430 7432 7437 7438 7439 7440 Function Quad 2-input NAND gates Quad 2-input NAND gates (open collector) Quad 2-input NOR gates Quad 2-input NOR gates (open collector) Hex inverters Hex inverters (open collector) Hex inverter buffer-driver Hex buffer-drivers Quad 2-input AND gates Quad 2-inputAND gates (open collector) Triple 3-input NAND gates Triple 3-input AND gates Triple 3-input NAND gates (open collector) Dual Schmitt triggers Hex Schmitt triggers Hex inverter buffer-drivers Hex buffer-drivers Dual 4-input NAND gates Dual 4-input AND gates Dual 4-input NAND gates (open collector) Expandable dual 4-input NOR gates Dual 4-input NOR gates Quad 2-input TTL-MOS interface NAND gates Triple 3-input NOR gates Quad 2-input NOR buffer 8-input NAND gate Quad 2-input OR gates Quad 2-input NAND buffers Quad 2-input NAND buffers (open collector) Quad 2-input NAND buffers (open collector) Dual 4-input NAND buffers Number 7441 7442 7443 7444 7445 7446 7447 7448 7450 7451 7452 7453 7454 7455 7459. 7460 7461 7462 7464 7465 7470 7472 7473 7474 7475 Function BCD-to-decimal decoder-Nixie driver BCD-to-decimal decoder Excess 3-to-decimal decoder Excess Gray-to-decimal BCD-to-decimal decoder-driver BCD-to-seven segment decoder-drivers (30-V output) BCD-to-seven segment decoder-drivers (15-V output) BCD-to-seven segment decoder-drivers Expandable dual 2-input 2-wide ANDOR-INVERT gates Dual 2-input 2-wide AND-OR-INVERT gates Expandable 2-input 4-wide AND-OR gates Expandable 2-input 4-wide AND-OR-INVERT gates 2-input 4-wide AND-OR-INVERT gates Expandable 4-input 2-wide AND-OR-INVERT gates Dual 2-3 input 2-wide AND-OR-INVERT gates Dual 4-input expanders Triple 3-input expanders 2-2-3-3 input 4-wide expanders 2-2-3-4 input 4-wide AND-OR-INVERT gates 4-wide AND-OR-INVERT gates (open collector) Edge-triggered JK flip-flop JK master-slave flip-flop Dual JK master-slave flip-flop Dual D flip-flop Quad latch Digital Principles and Applications Number 7476 7480 7482 7483 7485 7486 7489 7490 17491 7492 7493 7494 7495 7496 74100 74104 74105 74107 74109 74116 74121 74122 74123 74125 74126 74132 74136 74141 74142 74145 74147 74148 74150 74151 74152 74153 74154 74155 74156 74157 74160 74161 Function Dual JK master-slave flip-flop Gates full adder 2-bit binary full adder 4-bit binary full adder 4-bit magnitude comparator Quad EXCLUSIVE-OR gate 64-bit random-access read-write memory Decade counter 8-bi c shift register Divide-by-12 counter 4-bit binary counter 4-bit shift register 4-bit right-shift-left-shift register 5-bit parallel-in-parallel-out shift register 4-bit bistable latch JK master-slave flip-flop JK master-slave flip-flop Dual JK master-slave flip-flop Dual JK positive-edge-triggered flip-flop Dual 4-bit latches with clear Monostable multivibrator Monostable multivibrator with clear Monostable multivibrator Three-state quad bus buffer Three-state quad bus buffer Quad Schmitt trigger Quad 2-input EXCLUSIVE-OR gate BCD-to-decimal decoder-driver BCD counter-latch-driver BCD-to-decimal decoder-driver 10/4 priority encoder Priority encoder 16-line-to-l-line multiplexer 8-Channel digital multiplexer 8-Channel data selector-multiplexer Dual 4/1 multiplexer 4-line-to-16-line decoder-demultiplexer Dual 2/4 demultiplexer Dual 2/4 demultiplexer Quad 2/ 1 data selector Decade counter with asynchronous clear Synchronoi.;s 4-bit counter Number Function 74162 Synchronous 4-bit counter 74163 Synchronous 4-bit counter 74164 8-bit serial shift register 74165 Parallel-load 8-bit serial shift register 74166 8-bit shift register 74173 4-bit three-state register 74174 Hex F flip-flop with clear 74175 Quad D flip-flop with clear 74176 35-MHz presettable decade counter 74177 35-MHz presettable binary counter 74179 4-bit parallel-access shift register 74180 8-bit odd-even parity generator-checker 74181 Arithmetic-logic unit 74182 Look-ahead carry generator 74184 BCD-to-binary converter 74185 Binary-to-BCD converter 74189 Three-state 64-bit random-access memory 74190 Up-down decade counter 74191 Synchronous binary up-down counter 74192 Binary up-down counter 74193 Binary up-down counter 74194 4-bit directional shift register 74195 4-bit parallel-access shift register 74196 Presettable decade counter 74197 Presettable binary counter 74198 8-bit shift register 74199 8-bit shift register 74221 Dual one-shot Schmitt trigger 74251 Three-state 8-channel multiplexer 74259 8-bit addressable latch 74276 Quad JK flip-flop 74279 Quad debouncer 74283 4-bit binary full adder with fast carry 74284 Three-state 4-bit multiplexer 74285 Three-state 4-bit multiplexer 74365 Three-state hex buffers 74366 Three-state hex buffers 74367 Three-state hex buffers 74368 Three-state hex buffers 74390 Individual clocks with flip-flops 74393 Dual 4-bit binary counter 3: TTLDevices TTL CIRCUITS 74LS00 74LS04 14 7407/7417 14 74LS08 14 74LS10 14 14 2 13 2 13 2 13 2 13 2 13 3 12 3 12 3 12 3 12 3 12 ll 4 ll 4 ll 4 11 4 ll 4 5 10 5 IO 5 10 5 10 5 10 6 9 6 9 6 9 6 9 6 9 7 8 7 8 7 8 8 7 8 74LS1 I 74LSl2 14 2 13 74LSl3 14 2 13 74LS14 14 74LS20 14 14 2 13 2 13 2 13 12 3 12 3 12 3 12 3 12 3 4 II 4 ll 4 II 4 ll 4 II 5 10 5 10 5 IO 5 IO 5 10 6 9 6 9 6 9 6 9 6 9 7 8 7 8 7 8 7 8 7 8 74LS21 74LS27 14 74LS30 14 74LS32 14 74LS37 14 14 2 13 2 13 2 13 2 13 2 13 3 12 3 12 3 12 3 12 3 12 4 ll 4 II 4 II 4 II 4 ll 5 IO 5 lO 5 10 5 10 5 10 6 9 6 9 6 9 6 9 6 9 7 8 7 8 7 8 7 8 7 8 74LS38 74LS86 14 74LS125 14 74LS266 74LSl26 14 2 13 2 13 2 13 3 12 3 12 3 4 II 4 II 4 5 IO 5 10 6 9 6 7 8 7 14 14 2 13 12 3 ll 4 5 IO 9 6 8 7 2 13 12 3 12 II 4 II 5 IO 5 IO 9 6 9 6 9 8 7 8 7 8 Appendix 4: CMOS Devices 74HCOO Series Part No. Pins Function 74HCOO 74HC02 74HC04 74HC08 74HC14 74HC20 74HC30 74HC32 74HC42 74HC74 74HC85 74HC123 74HCl32 74HC138 74HC139 74HC154 74HC161 74HC163 74HC164 74HC165 74HC174 74HC175 74HCI91 74HC192 74HC193 74HC22I 74HC240 74HC244 14 14 14 14 14 14 14 14 16 14 16 16 14 16 16 S-24 16 16 14 16 16 16 16 16 16 16 20 20 Quad 2-input NAND gate Quad 2-input NOR gate Hex inverter (buffered) Quad 2-input AND gate Hex·inverting Schmitt trigger Dual 4-input NAND gate 8-input NAND gate Quad 2-input ORgate BCD-to-decimal decoder Dual D flip-flopwith preset and clear 4-bit magnitude comparator Dual monostable multivibrator Quad 2-i.nput NAND Schmitt trigger 3- to 8-line decoder Expandable dual 2- to 4-line decoder 4~ to 16-line decoder (use 24SLP socket) Synchronous binary counter Synchronous binary counter 8-bitserial in-parallel out shift register s~bitparallel in'-'serial out shift register Hex D Flip-Flop with dear Quad Dtype flip-flop with clear Up-downbinary counter Synchronous decade up-down counter Synchronous binary up-down counter Dual monostable multivibrator Inverting octal tri-state buffer Octa.I tri-state buffer _ _ _ _ _ _ _ _ _ _ _ _ _ _A_p_p_e_nd_ix_4_:_C_M_O_S_D_e_vic_e_s _ _ _ _ _ _ _ _ _ _ _ _ _ _ 74HCOO Series PartNo. Pins 74HC245 20 16 20 16 20 20 16 14 20 20 20 16 20 20 20 74HC373 74HC374 74HC390 74HC393 74HC541 74HC573 74HC574 74HC595 74HC688 74HC942 74HC943 74HC40l7 74HC4020 74HC4040 74HC4046 74:EI!---JEN 3-state output. Output with more than usual output capability (symbol is oriented in the direction of signal flow). Enable input When at its internal I-state, all outputs are enabled. When at its internal 0-state, open-collector, open-emitter, and three-state outputs are at external high-impedance state, and all other outputs (i.e., totem-poles) are at the internal 0-state. J,K,R, S, T --JD Usual meanings associated with flip-flops (e.g., R Data input to a storage element equivalent to: reset, T = toggle). 1=j ! Shift right (left) inputs, m = l, 2, 3, etc. lfm = 1, it is usually not shown. Binary grouping, mis highest power of 2. Produces a number equal to the sum of the weights of the active inputs. Input line grouping ... indicates two or more terminals used to implement a single logic input, e.g., differential inputs. + I 1 + Digital Principles and Applications It is particularly important to note that a D input is always the data input of a storage element. At its internal 1 state, the D input sets the storage element to its 1 state, and at its internal Ostate it resets the storage element to its O state. The binary grouping symbol is explained more fully in a later section. Binary-weighted inputs are arranged in order and the binary weights of the least-significant and the most-significant lines are indicated by numbers. In this document weights of input and output lines will be represented by powers of 2 usually only when the binary grouping symbol is used, otherwise decimal numbers will be used. The grouped inputs generate an internal number on which a mathematical function can be performed or that can be an identifying number for dependency notation. This number is the sum of the weights (I, 2, 4 ... , 2n) of those inputs standing at their 1 states. A frequent use is in addresses for memories. · Reversed in direction, the binary grouping symbol can be used with outputs. The concept is analogous to that for the inputs, and the weighted outputs will indicate the internal number assumed to be developed within the circuit. Appendix 8: Pinout Diagrams IA 18 2A 28 3A 38 4A 48 (I) IA IY (2) 18 (4) IC 2Y (5) (9) 2A 28 3Y (10) (12) 2C 3A (13) 38 JC Positive logic: Y"'" AB 00 (1) & IA (2) (13) 1B ()) JC (4) ID (5) 2A (9) 2B (10) 1C 2D (II) (I) 18 (4) IC (5) 2A (9) 28 (10) 2C (12) 3A (13) 38 JC 13 10 1B IA JC 2A 2A JA 28 4A 2C 5A (I) IY (13) lA (3) 2A (6) 2Y 38 Positive logic: Y ""A 4A (9) SA JC 04 (8) (11) (13) (3) (4) (5) (9) (10) (II) Positive logic: Y= A+B+C 27 18 2A 28 JA (5) (JO) JJ 3A 38 4A 6A JY 4B IB 2A 28 JA JB 4A 48 (I) IA JY (2) lB JC (4) 2Y (5) 2A (9) 28 (10) 2C (12) JA (13) JB 3C Positive logic: Y"' AB 08 (I) lY (4) (5) (9) (IO) (12) (13) Positive logic: Y "" A 14 Positive logic: Y ""ABC & ~l (2) Positive logic: Y"'" A+B 32 11 lA >I (2) lA (12) (4) (I) (I) & (2) JA 6A IA (2) Positive logic: Y"'" ABCD Positive logic: Y "" ABC IA &JJ & IA (2) (13) 1B (3) JC (4) JD (5) 2A (9) 28 (10) 2C (II) 2D Positive logic: Y = ABC 12 !ll IA (2) IB (4) 2A (5) 28 (9) 3A (10) JB (12) 4A (13) 48 Positive logic: Y"' ABCD 20 (I} &> IY (2) (4) (5) (9) (10) (12) (13) Positive logic: Y ""AB 37 Digital Principles and Applications IA (I) &1> COMP (2) :}p 38 PO Pl P2 P3 PQ > 48 QO QI Q2 Q3 IB 2A 2B 3A Positive logic: Y = AB 38 (6) IQ IQ PQ } (7) P Y) = G3 + E3G2 + E3E2GI + E3E2ElGO Ground pin 4 (after disconnecting from +5 V) and connect pin 3 to +5 V (after disconnecting from ground). 4.33 256 4.35 1100 4.37 The PROM. A ~ B C D ~-~· ~ Digital Principles and Applications 4.39 The PAL circuit. A B C D 4.41 3; 9; 15 (illegal) Chapter 5 5.1 5.3 5.5 5.7 5.9 5.11 5.13 5.15 5.17 5.19 5.21 5.23 5.25 01110000 a. 1 b. 2 C. 3 a. 188 b. 255 131,072 I 00001100 1010010100000 011010 101.111011110 504.771 a.257 b. 15.331 a. 1110 0101 b. 101101001101 12,121 a.0000 b. 0100 a. 011 0111 b. 101 0111 d.4 C. C. C. C. 123.55 0111 1010 1111 0100 1010 110 0110 d. 1111 d. 111 1001 Appendix 9: Answers to Selected Odd-Numbered Problems Alphanumeric Hex contents 2000 G 2001 C7 4F 5.27 Address 5.29 5.31 5.33 5.35 5 .3 7 5.39 2002 0 0 2003 D 2004 B C2 2005 y D9 45 4F C4 2006 E 3694 0001 0001 a, b, and d e 11100001111 Five for both (a) and (b) Chapter 6 6.1 6.3 6.5 6.7 6.9 6.11 6.13 a. 12 8 b. 1\ C. 10 16 0000 0101 0000 1000 0001 1000 a. 0001 0111 c. 1011 1000 b. 0111 1011 a. DCH b.BAH C. 36H a. 0100 1110 b. 1110 1001 C. 1010 0110 C. a. 0010 1101 b. 0101 1001 + 0011 1000 + 1101 1110 0110 0101 d. 1110 1011 d. 02H d. 1000 0111 0100 0011 + 1001 1110 1110 0001 0011 0111 6.15 02CBH, 0000 0010 1100 lOll 6.17 Binary 0101 0011, or decimal 83 6.19 G0 = 1.1 = 1, G1 = 1.0 = 0, G2 = 0.0 = 0, G3 = 1.1 = 1. po= l + 1 = 1, pl= 1 + 0 = 1, p2 = 0 + 0 = 0, p3 = 1 + 1 = 1 and c_l = 0. Substituting these in corresponding equations C0 = 1, C 1 = 1, C2 = 0, C 3 = 1. Using Si= GiEB Pi EB ci-1 so= 0, SI= 0, s2 = 1, s3 = 0. Final result: C3S3S2S1S0 = 10100 6.21 Substitute M = 0 and S3 •• S0 = 0110, Cm= 0, A= 1101 and B = 0111. Chapter 7 7.1 7.3 7.5 7.7 7.9 a. 100 ns b. 167 ns 13.3 MHz 0.45/4.05 3.5 MHz plus or minus 28 Hz C. CLK 1.33 µs __fl_flJL_ 11 Q 11 15 MHz II _J_r-:L__jt 7.5 MHz -11-td Digital Principles and Applications 7.11 2.5 A 1.7 0 0.9 3.4 B 0 0.2 +Vee C 0 7.15 48 kHz. t 1 = 13 µs, t2 = 7.8 µs 7.17 33.3 percent, 37.5 percent 7.19 RA+ RB= 15 kQ. RA= 3.75 kQ, RB= 11.25 kQ 7.21 3.88 ms 7.23 0.136µF 7.25 l-1 ms-l Input l-1.1 ms-, 7.27 Connect as in Example 7.8. 21.3 nF. 7.29 a. -1 b. f-1000 µs -l f-200 µs ~ _n_n_n_ru-ui_ __nrm J!--750µs-lu -l f-750 µs 7.31 Connect A1 to GND and apply input to B2 • C = 44.6 nF. 7.33 Same as Prob. 7.29. 7.35 7.37 LetR 1 =R 2 =lkQ. a. t 1 = 2.5 µs, t2 = 7.5 µs, C 1 = 7500 pF, C2 = 22,500 pF. b. ti = t2 = 1 µs, cl = c2 = 3000 pF. Appendix 9: Answers to Selected Odd-Numbered Problems Chapter 8 8.3 Q Q 8.5 a. C b. G 8.7 When the clock is low, the flip-flop is insensitive to levels on either R or S input. (Only first case is shown here.) S=O S=O CLOCK CLOCK R=O R=O 8.9 The Rand S inputs do not need to be held static while the clock is high. 8.11 Use negative-edge-triggering. t0 : S is low, R is high t 1: Sis high, R is low t2 and after: S is low, R is high. After t2 , both R and Scan be low. 8.13 Low 8.15 a. 5 ns b. 10 ns c. 15 ns 8.17 8.19 Clock period= 1 µs. Period of Q = 2 µs (f = 500 kHz) Digital Principles and Applications 8.21 +Vee 7 2 15 4 11 9 125 kHz 6 500 kHz 16 Pin 5 to +Vee Pin 13 to GND 12 8 3 8.23 The pulse symbol shows that the flip-flop is pulse-triggered. 8.25 R s Q +Vee R R 8.27 (a) Q,,+ 1 = BQ,, + AQ,; (b) l0 11 00~01 01_.,.~ll 00 I0 8.29 This is a modulo-3 counter with state sequence 00--+01-+ 10--+00 ... and co1Tesponding output, Y changes as 1--+0-+0-+ 1... 8.31 r-----------------------------------1 I I I I I T : I I I I I s Q R Q ,___,_.__,~ Q ____________________________________ ' I I I I Jl Q _ _ _ _ _ _ _ _ _A_pp_e_n_dix_9_:_An_.s_w_ei_s_to_S_e_le_ct_ed_O_d_d_-N_u_m_be_re_d_P_ro_b_le_m_s _ _ _ _ _ _ _ _ Chapter 9 9 .1 a. 6 b. 6 9.3 See Fig. 9.1 9.5 C. 4 A B C D CLOCKnn..rLil_ Djl_____JL 9.7 ABC DEF G H MSBjirst A=B 9.9 a. 8 µs b. 1.6 µs 9.11 16.7 MHz 9.13 a. R = 1, S = 0, Q = 0 9.15 b. R = 0, S = 1, Q = 1 Xl-----i ::::0---lS Control Clock--------------~ 9.17 MSB first, shift/load is low, ABCD EFGH= 1011 1110. CLOCK _ _ _~ INHIBIT L_j QB Qc QD QE QF " :o ------,,------ ---------------! ------! QG QH -------- t Stored number Q "Typical" ~ Digital Principles and Applications MSB first, shift/load is high. CLOCK QB --- Qc~~~-'r---L_J QD r---L_J QE~~~~~r--l___j QF -------------' QG~~~~~~~~~~ QH -----------------' CLOCK INHIBIT t Stored number 9.19 Same as 9.15. 9.21 MODE~ CLOCK l =CLOCK2 ~ I I I I I r+-, I I I I D, DATA INPUT _J : I : : : : I I I I I I I I ___s-i_LJ_J_ Q C _ _,_:_ _,Jl.__~~-- QD l • : ! :ri.__;....--1 Q I I l --'--- A --'--'----'-'- - ' 9.23 For alternate ls and Os, replace feedback with: toAB (a) Appendix 9: Answers to Selected Odd-Numbered Problems Include a power-on-CLEAR circuit like: +V C lOkQ to CLEAR lµFJ(b) This will CLEAR all flip-flops to zeros when power is first applied. 9 .25 Decoder output Y =R'S' Clock Serial in = T' Q R s T Y= R'S' 0 1 2 3 4 5 0 l 1 0 1 0 0 1 0 l 1 0 l 1 0 1 0 0 1 0 0 1 0 1 1 0 1 0 0 1 0 0 1 0 1 1 0 1 0 0 l 0 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 1 0 repeats 6 7 8 9 Chapter 10 10.1 J Clock SLJ1__f1_fl B A ~ B _ __, Clock K A K A 7i Clock B A 0 0 0 0 0 2 I 3 0 0 0 B 10.3 4MHz 10.5 Difficult to get clock and first few flip-flop outputs on the same page with the last few flip-flop outputs. 10. 7 Same as Fig. 10.3 where period of QB is 2 µs. 10.9 Sixteen4-inputNAND-gates_with inputs, ABC D,AB CD, ABC D,ABC D, A BCD, ABCD, ABCD,ABCD, ABCD,ABCD, ABCD,ABCD, ABCD,ABCD, ABCD, andABCD. i~"O" i~ D~ "!" ---------- 1§=)-"15' D, Digital Principles and Applications 10.11 CLK A A B C i,,,, c ,,,, B ,,,1 :1l i ji II II ABC Jl II 11 11 ABC ___J1 1: :: _,,,ii!----'"1-----':"'"": -ABC - - - -n__J]p'---'"1----',"-,-:: i.,,..,_ _ _ r-,:',-1----'+-----,,-:: ABC ABC II II ABC II ABC I ABC ~~~~~~~~~r-L__ 10.13 Same as Fig. 10.12c, except transitions occur on low-to-high clock. 10.15 As in Fig. 10.15. 10.17a.3 b.4 c.4 d.5 e.5 10.19 EP-"O" iD-"l" AP~D-"3" ~D-"4" iD-"5" ~ "1" _n__n_ _j__n__J_ "2" _J___n__J__ "O" "2" l l n l l "3" I I I I I I "5" _J____fl__J_ -i____n_ 5 6 "4" I I l 10.21 0 CLOCK CLOCK mod-3 mod-3 A1 Bi Az Bz I 2 3 4 7 8 0 _ _ _ _ _ _ _ _ _ _A_p_p_en_d_ix_9_:_A_ns_w_e_rs_t_o_s_el_ec_te_d_O_d_d_-N_u_m_b_e,.,_e_d_P_ro_b_le_f1'!_s_ _ _ _ _ _ _ _ _ 10.23 jp--, jp--, jp--, !P--, :p--, IP-· APIP--s fp--, [p-, B C 7 75 Clock count 0 2 3 4 5 6 0 2 3 4 5 6 7 8 9 Decode-counter outputs Decoding gates 10.25 Known gate time _FR_~ Unkn ownpu Ises Tocounter CLOCK input JlJUlJL 10.27 QD-------1 QC QB-------;__ __, to CLEAR QA QD QC ------r-, QB-----;____, to CLEAR QA 10.29 Like Prob. 10.27. 10.31 Draw the circuit from design equations DA =A®B, D8 =A'+B 7 g 9 (gi) 0 Digital Principles and Applications 10.33 Draw the circuit from design equationsJA=B + C', KA= B' + C'; J 8 =A, K8 =A; Jc=AB, Kc=A'B' 10.35 Design a modulo-6 counter (Section 10.7) with state sequence for three flip-flops CBA 000 -7 001 -7 010 -7 011 -7 100 -7 101 -7 000 ... Then draw circuit for output generating sequence as Y = C + A'B' 10.39 Reconnect the J input on flip-flop A to l5. 10.41 Mod-5 illegal states are 2(010), 5(101), and 7(111). AND gate will detect AB and force counter to CBA. Count 7 will progress to count 6(110). Mod-3 illegal state is 3(11), which will progress naturally to count 2(10). 10.43 ~-r--\~ "!" ·D~ ~=0-"3" i=0-"5" Chapter 11 11.1 1/0 11.3 0 0 11.5 11.7 Present State Present Input Next State Present Output B" An Xn Bn+1 A.+1 y JB KB JA KA 0 0 0 0 1 l 0 0 1 1 1 0 1 0 0 0 0 0 0 0 0 1 0 1 X X X X l X X 0 X 1 0 X l 0 1 1 0 1 l 0 0 0 0 1 0 X l 0 l 0 l 0 1 X 0 n X X _ _ _ _ _ _ _ _ _A_p_p_e_nd_ix_9_:_An_s_w_ers_to_S_e_le_ct_e_d_O_dd_-N_u_m_b_ere_d_P_ro_bl_em_s_ _ _ _ _ _ _ _ _ (j) 11.9 I Bt4 I II 00 01 11 10 0 0 0 X X 1 0 I1 xi X ¥ I II 00 01 11 10 0 Ix X 1 xi 1 X X 0 X X II 00 01 11 10 0 0 X X X I Il X X xi J A =X I II I 00 01 11 10 0 Ix l l xi 1 X 0 0 X II 00 01 11 10 0 0 0 11 xi l 0 0 0 X Y=XB11 Compared to solution given in Section 11.4, this requires one AND gates less for JA input. 11.11 Two for Mealy and three for Moore circuit. 11.13 Two flip-flops (B. and A) are required. Three states assigned as (BA) 00, 01, 11 representing no, 1si, 2°d bit detection respectively. Then, D 8 =X'B'A DA =X'B' Output Y=X'BA 11.15 3 to 8 decoder Bn A" X _ _ f _[~ x;, Bn+l A11+1 y11 000 0 0 0 001 0 l 0 010 0 0 0 011 l 0 0 100 0 0 l 101 l 0 0 110 X X X lll X X X 8 x3ROM y ' Bn / Bn+l D .