Mc Graw Hill Education 大学计算机教育国外著名教材系列（影印版）

INTRODUCTION TO LOGIC DESIGN

SECOND EDITION

逻辑设计基础

（第2版）

Alan B.Marcovitz 著

清华大学出版社

大学计算机教育国外著名教材系列（影印版）

Introduction to Logic Design

Second Edition

逻辑设计基础

（第2版）

Alan B. Marcovitz 著

Florida Atlantic University

清华大学出版社

北京

Alan B. Marcovitz
Introduction to Logic Design, Second Edition
EISBN: 0-07-295176-1

北京市版权局著作权合同登记号　图字 01-2005-6158

图书在版编目(CIP)数据

逻辑设计基础：第 2 版 /（美）马科维兹（Marcovitz, A. B.）著. —影印本. —北京：清华大学出版社，2006.9
（大学计算机教育国外著名教材系列）
ISBN 7-302-13553-3

Ⅰ. 逻…　Ⅱ. 马…　Ⅲ. 电子计算机－逻辑设计－高等学校－教材－英文　Ⅳ. TP302.2

中国版本图书馆 CIP 数据核字（2006）第 086802 号

出 版 者：清华大学出版社　　地　　址：北京清华大学学研大厦
http://www.tup.com.cn　　邮　　编：100084
社总机：010-6277 0175　　客户服务：010-6277 6969
印 刷 者：北京市昌平环球印刷厂
装 订 者：三河市李旗庄少明装订厂
发 行 者：新华书店总店北京发行所
开　　本：185×230　印张：42
版　　次：2006 年 9 月第 1 版　2006 年 9 月第 1 次印刷
书　　号：ISBN 7-302-13553-3/TP・8501
印　　数：1~3000
定　　价：59.00 元

出 版 说 明

进入21世纪，世界各国的经济、科技以及综合国力的竞争将更加激烈。竞争的中心无疑是对人才的竞争。谁拥有大量高素质的人才，谁就能在竞争中取得优势。高等教育，作为培养高素质人才的事业，必然受到高度重视。目前我国高等教育的教材更新较慢，为了加快教材的更新频率，教育部正在大力促进我国高校采用国外原版教材。

清华大学出版社从1996年开始，与国外著名出版公司合作，影印出版了“大学计算机教育丛书（影印版）”等一系列引进图书，受到国内读者的欢迎和支持。跨入21世纪，我们本着为我国高等教育教材建设服务的初衷，在已有的基础上，进一步扩大选题内容，改变图书开本尺寸，一如既往地请有关专家挑选适用于我国高校本科及研究生计算机教育的国外经典教材或著名教材，组成本套“大学计算机教育国外著名教材系列（影印版）”，以飨读者。深切期盼读者及时将使用本系列教材的效果和意见反馈给我们。更希望国内专家、教授积极向我们推荐国外计算机教育的优秀教材，以利我们把“大学计算机教育国外著名教材系列（影印版）”做得更好，更适合高校师生的需要。

清华大学出版社

PREFACE

This book is intended as an introductory logic design book for students in computer science, computer engineering, and electrical engineering. It has no prerequisites, although the maturity attained through an introduction to engineering course or a first programming course would be helpful.

The book stresses fundamentals. It teaches through a large number of examples. The philosophy of the author is that the only way to learn logic design is to do a large number of design problems. Thus, in addition to the numerous examples in the body of the text, each chapter has a set of Solved Problems, that is, problems and their solutions, a large set of Exercises (with answers to selected exercises in Appendix B), and a Chapter Test (with answers in Appendix C). In addition, there are a set of laboratory experiments that tie the theory to the real world. Appendix A provides the background to do these experiments with a standard hardware laboratory (chips, switches, lights, and wires), a breadboard simulator (for the PC or Macintosh), and two schematic capture tools. The course can be taught without the laboratory, but the student will benefit significantly from the addition of 8 to 10 selected experiments.

Although computer-aided tools are widely used for the design of large systems, the student must first understand the basics. The basics provide more than enough material for a first course. The schematic capture laboratory exercises and a section on Hardware Design Languages in Chapter 8 provide some material for a transition to a second course based on one of the computer-aided tool sets.

Chapter 1 gives a brief overview of number systems as it applies to the material of this book. (Those students who have studied this in an earlier course can skip to Section 1.2.) It then discusses the steps in the design process for combinational systems and the development of truth tables.

Chapter 2 introduces switching algebra and the implementation of switching functions using common gates—AND, OR, NOT, NAND, NOR, Exclusive-OR, and Exclusive-NOR. We are only concerned with the logic behavior of the gates, not the electronic implementation.

Chapter 3 deals with simplification using the Karnaugh map. It provides methods for solving problems (up to six variables) with both single and multiple outputs.

Chapter 4 introduces two algorithmic methods for solving combinational problems—the Quine-McCluskey method and Iterated Consensus. Both provide all of the prime implicants of a function or set of

functions, and then use the same tabular method to find minimum sum of products solutions.

Chapter 5 is concerned with the design of larger combinational systems. It introduces a number of commercially available larger devices, including adders, comparators, decoders, encoders and priority encoders, and multiplexers. That is followed by a discussion of the use of logic arrays—ROMs, PLAs, and PALs for the implementation of medium scale combinational systems. Finally, two larger systems are designed.

Chapter 6 introduces sequential systems. It starts by examining the behavior of latches and flip flops. It then discusses techniques to analyze the behavior of sequential systems.

Chapter 7 introduces the design process for sequential systems. The special case of counters is studied next. Finally, the solution of word problems, developing the state table or state diagram from a verbal description of the problem is presented in detail.

Chapter 8 looks at larger sequential systems. It starts by examining the design of shift registers and counters. Then, PLDs are presented. Three techniques that are useful in the design of more complex systems—ASM diagrams, one-shot encoding, and HDLs—are discussed next. Finally, two examples of larger systems are presented.

Chapter 9 deals with state reduction and state assignment issues. First, a tabular approach for state reduction is presented. Then partitions are utilized both for state reduction and for achieving a state assignment that will utilize less combinational logic.

A feature of this text is the Solved Problems. Each chapter has a large number of problems, illustrating the techniques developed in the body of the text, followed by a detailed solution of each problem. Students are urged to solve each problem (without looking at the solution) and then compare their solution with the one shown.

Each chapter contains a large set of exercises. Answers to a selection of these are contained in Appendix B. Solutions will be made available to instructors through the Web. In addition, each chapter concludes with a Chapter Test; answers are given in Appendix C.

Another unique feature of the book is the laboratory exercises, included in Appendix A. Four platforms are presented—a hardware based Logic Lab (using chips, wires, etc.); a hardware lab simulator that allows the student to "connect" wires on the computer screen; and two circuit capture programs, LogicWorks 4 and Altera Max+plus II. Enough information is provided about each to allow the student to perform a variety of experiments. A set of 26 laboratory exercises are presented. Several of these have options, to allow the instructor to change the details from one term to the next.

We teach this material as a four-credit course that includes an average of 3 1/2 hours per week of lecture, plus, typically, eight laboratory exercises. (The lab is unscheduled; it is manned by Graduate

Assistants 40 hours per week; they grade the labs.) In that course we cover

Chapter 1: all of it

Chapter 2: all but 2.11

Chapter 3: all of it

Chapter 4: if time permits at the end of the semester

Chapter 5: all but 5.8. However, there is a graded design problem based on that material (10 percent of the grade; students usually working in groups of 2 or 3).

Chapter 6: all of it

Chapter 7: all of it

Chapter 8: 8.1, 8.2, 8.3. We sometimes have a second project based on 8.7.

Chapter 9 and Chapter 4: We often have some time to look at one of these. We have never been able to cover both.

With less time, the coverage of Section 2.10 could be minimized. Section 3.5 is not needed for continuity; Section 3.6 is used somewhat in the discussion of PLAs in Section 5.7.2. Chapter 5 is not needed for anything else in the text, although many of the topics are useful to students elsewhere. The instructor can pick and choose among the topics. The *SR* and *T* flip flops could be omitted in Chapters 6 and 7. Sections 7.2 and 7.3 could be omitted without loss of continuity. As is the case for Chapter 5, the instructor can pick and choose among the topics of Chapter 8. With a limited amount of time, Section 9.1 could be covered. With more time, it could be skipped and state reduction taught using partitions (9.2 and 9.3).

ACKNOWLEDGMENTS

I want to thank my wife, Allyn, for her encouragement and for enduring endless hours when I was closeted in my office working on the manuscript. Several of my colleagues at Florida Atlantic University have read parts of the manuscript and have taught from earlier drafts. I wish to acknowledge especially Mohammad Ilyas, Imad Mahgoub, Oge Marques, Imad Jawhar, Abhi Pandya, and Shi Zhong for their help. In addition, I wish to express my appreciation to my chairs, Mohammad Ilyas, Roy Levow, and Borko Fuhrt who made assignments that allowed me to work on the book. Even more importantly, I want to thank my students who provided me with the impetus to write a more suitable text, who suffered through earlier drafts of the book, and who made many suggestions and corrections. I want to thank Visram Rathnam for his contributions to the section on Altera tools. The reviewers—

Michael McCool, University of Waterloo;

Pinaki Mazumder, University of Michigan;

Nick Phillips, Southern Illinois University;
Gary J. Minden, University of Kansas;
Daniel J. Tylavsky, Arizona State University;
Nadar I. Rafla, Boise State University;
Dan Stanzione, Clemson University;
Frank M. Candocia, Florida International University;
Lynn Stauffer, Sonoma State University;
Rajeev Barua, University of Maryland—

provided many useful comments and suggestions. The book is much better because of their efforts. Finally, the staff at McGraw-Hill, particularly Carlise Paulson, Melinda Dougharty, Jane Mohr, Betsy Jones, Barbara Somogyi, Rick Noel, Sandy Ludovissy, Audrey Reiter, and Dawn Bercier have been indispensable in producing the final product, as has Michael Bohrer-Clancy at Interactive Composition Corporation.

Alan Marcovitz

BRIEF CONTENTS

CONTENTS

Chapter 1

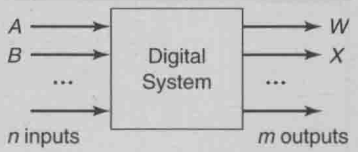
Introduction

This book concerns the design of digital systems, a process often referred to as logic design. A digital system is one in which all of the signals are represented by discrete values. Computers and calculators are obvious examples, but most electronic systems contain a large amount of digital logic. Internally, digital systems usually operate with two-valued signals, which we will label 0 and 1. Although multi-valued systems have been built, two-valued systems are more reliable, and thus almost all digital systems use two-valued signals. Such a system, as shown in Figure 1.1, may have an arbitrary number of inputs ($A, B, \ldots$) and an arbitrary number of outputs ($W, X, \ldots$).

In addition to the data inputs shown, some circuits require a timing signal, called a clock (which is just another input signal that alternates between 0 and 1 at a regular rate). We will discuss the details of clock signals in Chapter 6.

A simple example of digital systems is shown in Example 1.1.

Figure 1.1 A digital system.



EXAMPLE 1.1

A system with three inputs, A, B, and C, and one output, Z, such that $Z = 1$ if and only if[1] two of the inputs are 1.

The inputs and outputs of a digital system represent real quantities. Sometimes, as in Example 1.1, these are naturally binary, that is, they take on one of two values. Other times, they may be multivalued. For example, an input may be a decimal digit or the output might be the letter grade for this course. Each must be represented by a set of binary digits (often called bits). This process is referred to as coding the inputs and outputs into binary. (We will discuss the details of this later.)

[1]The term *if and only if* is often abbreviated iff. It means that the output is 1 if the condition is met and is not 1 (which means it must be 0) if the condition is not met.

The physical manifestation of these binary quantities may be one of two voltages, for example, 0 volts or ground for logic 0 and 5 volts for logic 1, as in the laboratory implementations we will be discussing in Appendix A.1. It may also be a magnetic field in one direction or another (as on diskettes), a switch in the up or down position (for an input), or a light on or off (as an output). Except in the discussion of specific laboratory experiments and in the translation of verbal descriptions into more formal ones, the physical representation will be irrelevant in this text; we will be concerned with 0's and 1's.

Table 1.1 A truth table for Example 1.1.

A	*B*	*C*	*Z*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

We can describe the behavior of a digital system, such as that of Example 1.1, in tabular form. Since there are only eight possible input combinations, we can list all of them and what the output is for each. Such a table (referred to as a truth table) is shown in Table 1.1. We will leave the development of truth tables (including one similar to this) to later in the chapter.

Three other examples are given in Examples 1.2, 1.3, and 1.4.

EXAMPLE 1.2

A system with eight inputs, representing two 4-bit binary numbers, and one 5-bit output, representing the sum. (Each input number can range from 0 to 15; the output can range from 0 to 30.)

EXAMPLE 1.3

A system with one input, *A*, plus a clock, and one output, *Z*, which is 1 iff the input was one at the last three consecutive clock times.

EXAMPLE 1.4

A more complex example is a traffic controller. In the simplest case, there are just two streets, and the light is green on each street for a fixed period of time. It then goes to yellow for another fixed period and finally to red. There are no inputs to this system other than the clock. There are six outputs, one for each color in each direction. (Each output may control multiple bulbs.) Traffic controllers may have many more outputs, if, for example, there are left-turn signals. Also, there may be several inputs to indicate when there are vehicles waiting at a red signal or passing a green one.

The first two examples are *combinational*, that is, the output depends only on the present value of the input. In Example 1.1, if we know the value of *A*, *B*, and *C* right now, we can determine what *Z* is now.[2] Examples 1.3 and 1.4 are *sequential*, that is, they require *memory*, since we need to know something about inputs at an earlier time (previous clock times).

We will concentrate on combinational systems in the first half of the book and leave the discussion about sequential systems until later. As we

[2]In a real system, there is a small amount of delay between the input and output, that is, if the input changes at some point in time, the output changes a little after that. The time frame is typically in the nanosecond (10^{-9} sec) range. We will ignore those delays almost all of the time, but we will return to that issue in Chapter 5.

will see, sequential systems are composed of two parts, memory and combinational logic. Thus, we need to be able to design combinational systems before we can begin designing sequential ones.

A word of caution about natural language in general, and English in particular, is in order. English is not a very precise language. The examples given above leave some room for interpretation. In Example 1.1, is the output to be 1 if all three of the inputs are 1, or only if exactly two inputs are 1? One could interpret the statement either way. When we wrote the truth table, we had to decide; we interpreted "two" as "two or more" and thus made the output 1 when all three inputs were 1. (In problems in this text, we will try to be as precise as possible, but even then, different people may read the problem statement in different ways.)

The bottom line is that we need a more precise description of logic systems. We will develop that for combinational systems in the first two chapters and for sequential systems in Chapter 6.

1.1 A BRIEF REVIEW OF NUMBER SYSTEMS

This section gives an introduction to some topics in number systems, primarily those needed to understand the material in the remainder of the book. We will only deal with integers. If this is familiar material from another course, skip to Section 1.2 (page 19).

Integers are normally written using a positional number system, where each digit represents the coefficient in a power series

$$N = a_{n-1}r^{n-1} + a_{n-2}r^{n-2} + \cdots + a_2r^2 + a_1r + a_0$$

where n is the number of digits, r is the radix or base, and the a_i are the coefficients, where each is an integer in the range

$$0 \leq a_i < r$$

For decimal, $r = 10$, and the a's are in the range 0 to 9. For binary, $r = 2$, and the a's are all either 0 or 1. Other commonly used notations in computer documentation are octal, $r = 8$, and hexadecimal, $r = 16$. In binary, the digits are usually referred to as *bits*, a contraction for *b*inary dig*its*.

The decimal number 7642 (sometimes written 7642_{10} to emphasize that it is radix 10, that is, decimal) thus stands for

$$7642_{10} = 7 \times 10^3 + 6 \times 10^2 + 4 \times 10 + 2$$

and the binary number

$$\begin{aligned} 101111_2 &= 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2 + 1 \\ &= 32 + 8 + 4 + 2 + 1 = 47_{10} \end{aligned}$$

From this last example,[3] it is clear how to convert from binary to decimal; just evaluate the power series. To do that easily, it is useful to know the powers of 2, rather than compute them each time they are needed. (It would save a great deal of time and effort if at least the first ten powers of 2 were memorized; the first 20 are shown in the Table 1.2.)

Table 1.2 Powers of 2.

n	2^n	n	2^n
1	2	11	2,048
2	4	12	4,096
3	8	13	8,192
4	16	14	16,384
5	32	15	32,768
6	64	16	65,536
7	128	17	131,072
8	256	18	262,144
9	512	19	524,288
10	1,024	20	1,048,576

We will often be using the first 16 positive binary integers, and sometimes the first 32, as shown in the Table 1.3. (As in decimal, leading 0's are often left out, but we have shown the 4-bit number including leading 0's for the first 16.) When the size of the storage place for a positive binary number is specified, then leading 0's are added so as to obtain the correct number of bits.

Table 1.3 First 32 binary integers.

Decimal	Binary	4-bit	Decimal	Binary
0	0	0000	16	10000
1	1	0001	17	10001
2	10	0010	18	10010
3	11	0011	19	10011
4	100	0100	20	10100
5	101	0101	21	10101
6	110	0110	22	10110
7	111	0111	23	10111
8	1000	1000	24	11000
9	1001	1001	25	11001
10	1010	1010	26	11010
11	1011	1011	27	11011
12	1100	1100	28	11100
13	1101	1101	29	11101
14	1110	1110	30	11110
15	1111	1111	31	11111

Note that the number one less than 2^n consists of n 1's (for example, $2^4 - 1 = 1111 = 15$ and $2^5 - 1 = 11111 = 31$).

[3]Section 1.6, Solved Problems, contains additional examples of each of the types of problems discussed in this chapter. There is a section of Solved Problems in each of the chapters.

An n-bit number can represent the positive integers from 0 to $2^n - 1$. Thus, for example, 4-bit numbers have the range of 0 to 15, 8-bit numbers 0 to 255 and 16-bit numbers 0 to 65,535.

To convert from decimal to binary, we could evaluate the power series of the decimal number, by converting each digit to binary, that is

$$746 = 111 \times (1010)^{10} + 0100 \times 1010 + 0110$$

but that requires binary multiplication, which is rather time-consuming.

There are two straightforward algorithms using decimal arithmetic. First, we can subtract from the number the largest power of 2 less than that number and put a 1 in the corresponding position of the binary equivalent. We then repeat that with the remainder. A 0 is put in the position for those powers of 2 that are larger than the remainder.

EXAMPLE 1.5

For 746, $2^9 = 512$ is the largest power of 2 less than or equal to 746, and thus there is a 1 in the 2^9 (512) position. We then compute $746 - 512 = 234$. The next smaller power of 2 is $2^8 = 256$, but that is larger than 234 and thus, there is a 0 in the 2^8 position. Next, we compute $234 - 128 = 106$, putting a 1 in the 2^7 position. (Now, the binary number begins 101.) Continuing, we subtract 64 from 106, resulting in 42 and a 1 in the 2^6 position (and now the number begins with 1011). Since 42 is larger than 32, we have a 1 in the 2^5 position, and compute $42 - 32 = 10$. Since $2^4 = 16$ is greater than 10, there is a 0 in the 2^4 position. At this point, we can continue subtracting (8 next) or recognize that the binary equivalent of the remainder, 10, is 1010, giving

$$\begin{aligned} 746_{10} &= 1 \times 2^9 + 0 \times 2^8 + 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 \\ &\quad + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2 + 0 \\ &= 1011101010_2 \end{aligned}$$

The other approach is to divide the decimal number by 2 repeatedly. The remainder each time gives a digit of the binary answer, starting at the least significant bit (a_0). The remainder is then discarded and the process is repeated.

EXAMPLE 1.6

Converting 746 from decimal to binary, we compute

746/2 = 373 with a remainder of 0	0
373/2 = 186 with a remainder of 1	10
186/2 = 93 with a remainder of 0	010
93/2 = 46 with a remainder of 1	1010
46/2 = 23 with a remainder of 0	01010
23/2 = 11 with a remainder of 1	101010
11/2 = 5 with a remainder of 1	1101010
5/2 = 2 with a remainder of 1	11101010
2/2 = 1 with a remainder of 0	011101010
1/2 = 0 with a remainder of 1	1011101010