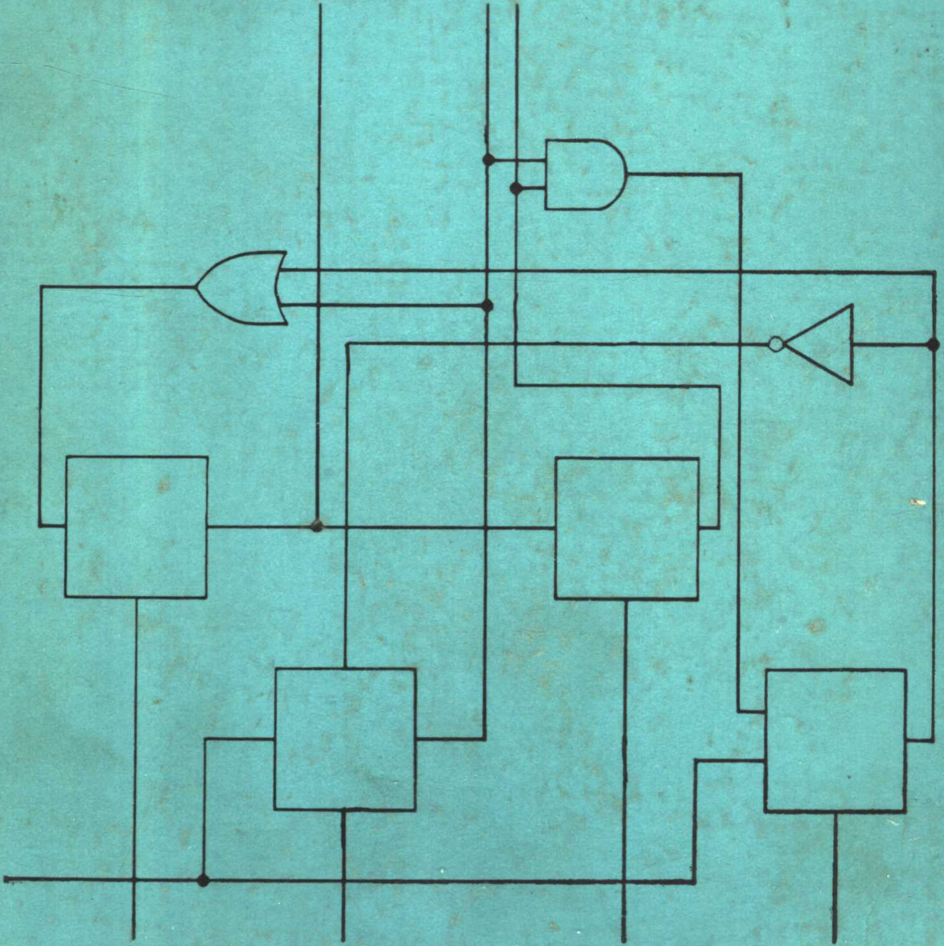
Logic Designer's Manual



JOHN D. LENK

Logic Designer's Manual

John D. Lenk
Consulting Technical Writer

Reston Publishing Company
A Prentice-Hall Company
Reston, Virginia

Library of Congress Cataloging in Publication Data

Lenk, John D
Logic designer's manual.

Includes index.
1. Logic circuits. 2. Computers—Circuits.
3. Electronic circuit design. I. Title.
TK7868.L6L46 621.3819'535 76-52957
ISBN 0-87909-450-8

10 9 8 7 6 5 4 3 2 1

Printed in the United States of America

Preface

The *Logic Designer's Manual* is an outgrowth of the author's popular *Handbook of Logic Circuits*. Most of today's logic design is accomplished with complete logic circuits, generally found in IC (integrated circuit) and/or plug-in module form. In prior years, it was necessary for the designer to implement such circuits as decoders, flip-flops, counters, registers, distributors, arithmetic units, and so on. Today, all these circuits are available in IC form, thus eliminating the need to make up circuits using basic logic gates. However, to use packaged logic circuits effectively, today's designer must be able to interconnect off-the-shelf logic circuits to form logic systems. The purpose of the *Logic Designer's Manual* is to fill that need.

This manual is written for logic IC users, rather than for designers of logic ICs. That is, the manual is written on the basis of using existing, commercial logic ICs to solve design and application problems. Typical users include design specialists who want to develop logic systems with available ICs, or technicians who must service logic equipment containing logic ICs. Experimenters and hobbyists can also make good use of this approach to logic IC applications.

There are two very common, although not necessarily accurate, concepts concerning logic ICs. First, it is assumed that the basic approach to logic design involves informing an IC manufacturer of design parameters and requirements for a particular logic system, and instructing the manufacturer to fabricate an IC (or group of ICs, or possibly a microprocessor) which meets these exact requirements. While this approach is satisfactory for some highly specialized logic systems, and particularly where cost is no factor, the approach may generally be wasteful and often unnecessary. On the other hand, it is often assumed that existing commercial logic ICs are limited in application, that such ICs are designed with only one or two uses in view.

There is a middle ground. Except for certain very special circuits, there are a number of commercial logic ICs that can be adapted to meet most logic circuit and system requirements. Also, new logic IC modules are being developed by various manufacturers. Likewise, although most off-the-shelf logic ICs are manufactured with certain specific uses in mind, these ICs are certainly not limited to only those uses. Thus, the approach found in this manual serves a two-fold purpose: (1) to acquaint the reader with logic ICs, in general, so that the user can select commercial units to meet his particular requirements, and (2) to show the reader the many other uses for existing logic ICs not found on the manufacturer's datasheet.

Chapter 1 is an introduction to logic design, which includes the basics of logic circuits, the logic symbols in common use throughout the industry, the basic principles of logic equations, and corresponding functions. The first chapter also provides detailed procedures for the design of both combinational and sequential networks. This chapter summarizes the entire subject of logic design, on the assumption that many readers will be students who are not familiar with all phases of the logic field.

Chapter 2 describes logic circuit elements available in IC form (decoders, distributors, counters, registers, and so on). This chapter describes what is available as well as how the circuits operate, and how they are used in basic systems.

The remainder of the text, Chapters 3 through 10, covers design applications of the basic circuits and elements discussed in Chapter 2. These chapters cover such subjects as data selectors, decoders, counters, registers, analog/digital and digital/analog converters, arithmetic units, memories, interface circuits, noise problems, and miscellaneous logic devices. The circuits described represent a cross-section of the entire logic field.

The author considers logic design to be an art rather than an exact science. He recognizes that there are many alternate solutions or designs to the problems described here. However, the designs presented in this manual are proven with time and will get the job done. To use the manual effectively, the reader is invited to study both the alphabetical index (at the end of the book) and the table of contents. In either or both lists, the reader will often find one or more designs listed that meet his exact needs. If not, the circuits described in this manual will at least provide a starting point for special design requirements.

The author has received much help from many organizations and individuals prominent in the field of logic design. He wishes to thank them all, and wants to express special thanks to the following: Cambridge Thermionic Corporation; Digital Equipment Corporation; Hewlett-Packard; Honeywell, Inc.; International Telephone & Telegraph; Litton Industries; Motorola Semiconductor Products, Inc.; Radio Corporation of America; and Texas Instruments Incorporated.

The author also wishes to express his appreciation to Mr. Joseph A. Labok of Los Angeles Valley College, and to Mr. Richard L. Castellucis of Southern Technical Institute for their help and encouragement.

John D. Lenk

Contents

1

Introduction to Logic Design

Today's logic designer must work with logic equations, circuits implemented by interconnecting basic logic gates, and complete logic circuits in IC module form. No matter what logic elements are used, many problems in logic design can be solved by working out the solution in equation form first, and then making circuits (or interconnecting IC elements) to match the equations. As an example, if a circuit is to be ***minimized*** (reduced to the least number of logic elements), the equipment is reduced to the simplest form "on paper," and the simplified equation is converted to a circuit.

Even when simplification is not involved, it is easier to write an equation than to wire a circuit. A good example of this is a logic module that has six inputs and one output, in which the output must be present when three (and only three) of the inputs are present. With such a problem, the equation is written to express the relationship of the six inputs to the output. Then the equation is converted to a corresponding circuit.

Today's designer must be familiar with logic equations (also known as logical algebra, Boolean algebra—after the English mathematician George Boole—computer algebra, or computer logic). The designer must be able to manipulate logical equations (usually to simplify them), and then to convert the equations into practical circuits. This task is greatly simplified if the designer can write an equation for a given circuit as a first step to improving the circuit. For example, it may be necessary to convert a circuit from positive logic to negative logic in order to accommodate an external circuit function.

In addition to a practical working knowledge of logical algebra, the designer must be familiar with the symbols used in logic circuits, the binary number system (typical to most logic systems), the basic logic circuit forms, mapping of logic circuits, and certain other problems common to all logic design. These subjects are discussed *in summary form* throughout this chapter.

1–1 DEFINING LOGICAL ALGEBRA

Conventional algebra is the symbolic expression for relationships of number variables. Logical algebra is a method of symbolically expressing the relationship between logic variables. Logical algebra differs from conventional algebra in two respects:

1. Arithmetic operations are not performed in logical algebra.

2. The symbols used in logical algebra (usually letters) do not represent numerical values.

Logical algebra is ideally suited to any system of intelligence based on *two opposite states*, such as "on" or "off." Thus, logical algebra is specifically suited to express the opening and closing of electrical switches, the presence or absence of electrical pulses, or the polarity or amplitude relationship of pulses. Logical algebra is also compatible with the *binary number system* (a two-state arithmetic system), and the various special logic codes discussed in Secs. 1–2 and 1–3.

1–2 BINARY NUMBER SYSTEM

In the binary system, all numbers can be maae up by using only ones and zeros, rather than zero through nine, as in the decimal system. Consequently, instead of requiring ten different values to represent one digit, logic circuits using the binary method need only two values for each digit. In logic circuits, these values are easily indicated by the presence or absence of a signal (or pulse), or by positive and negative signals, or even by two different voltage levels.

In binary, the value of each digit is based on 2, and the powers of 2. In a binary number, the extreme right-hand digit is multiplied by 1, the second-from-the-right digit is multiplied by 2, the third-from-the-right digit is multiplied by 4, and so on. This can be displayed as follows:

2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
256	128	64	32	16	8	4	2	1

In binary, if the digit is zero, its value is zero. If the digit is one (1), its value is determined by its position from the right. For example, to represent the number 77 in binary form, the following combination of zeros and ones is used:

256	128	64	32	16	8	4	2	1
0	0	1	0	0	1	1	0	1

$$64 + 0 + 0 + 8 + 4 + 0 + 1 = 77$$

which means 1001101 in pure binary form = 77.

1–2.1 Converting Decimal Numbers Into Binary Numbers

A decimal number can be converted into a binary number in two ways. The obvious way is to make up a chart showing the power of two, as has just been done, and to then count the necessary number of ones and zeros to make up the desired decimal number.

For example, assume that the number 33 is to be converted. The number 33 is more than 32 (sixth position from the right) but less than 64 (seventh position from the right). This means that you need a combination of six digits (ones or zeros—probably both).

Start with the sixth position, or 32. Since you want number 32, write a one in the sixth position. Then move to the fifth position, or 16. Thirty-two plus 16 is greater than the desired 33, so use a zero for the fifth position. The fourth position is 8, and 8 plus 32 is more than 33, so you use a 0 for the fourth position. The same is true of the third position, or 4, and the second position, or 2, so both of these positions use a 0. The first (right-hand) position is 1, and 1 plus the sixth position (or 32) makes the desired 33, so both of these positions require a 1. Thus, the pure binary equivalent of 33 is 100001. This is shown in Figure 1-1, along with some other examples for tabular conversion of decimal and binary numbers.

	Binary					
Decimal	2^5	2^4	2^3	2^2	2^1	2^0
	32	16	8	4	2	1
0	0	0	0	0	0	0
1	0	0	0	0	0	1
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	0	0	1	0	0
5	0	0	0	1	0	1
6	0	0	0	1	1	0
7	0	0	0	1	1	1
8	0	0	1	0	0	0
9	0	0	1	0	0	1
10	0	0	1	0	1	0
20	0	1	0	1	0	0
30	0	1	1	1	1	0
→ 33	1	0	0	0	0	1
40	1	0	1	0	0	0
47	1	0	1	1	1	1
50	1	1	0	0	1	0

$\boxed{33 = 1\ 0\ 0\ 0\ 0\ 1}$

Figure 1-1. Tabular conversion of decimal and binary numbers.

The alternate method for converting from decimal to binary is to divide the decimal number by two as many times as is necessary to lower the quotient to a number less than two (1 or 0), using the *remainders* for each step of division as the binary numbers. This is shown in Figure 1-2.

Successive dividers	Original number and dividends	Remainder (binary number)
2	33	1
2	16	0
2	8	0
2	4	0
2	2	0
2	1	1 — 1 0 0 0 0 1

$\boxed{33 = 1\ 0\ 0\ 0\ 0\ 1}$

Figure 1-2. Converting decimal numbers to binary numbers by division.

For example, again assume that 33 is to be converted. Thirty-three divided by 2 is 16, with a remainder of 1. This 1 is the right-hand or first-position digit (also known as the least significant digit, or LSD).

Sixteen divided by 2 is 8 with a remainder of 0. This 0 is the second-position digit.

Eight divided by 2 is 4 with a remainder of 0. This 0 is the third-position digit. Two divided by 2 is 1 with a remainder of 0. This 0 is the fourth-position digit.

One divided by 2 is considered as 0 (since the whole number 1 can not be divided by 2), and there is a remainder of 1. This 1 is the sixth position. The 1 is also the left-hand or last-position digit (known as the most significant digit, or MSD).

Thus, the binary count for the decimal number 33 is 100001.

1–2.2 Adding Binary Numbers

The rules for adding binary numbers are:

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 0$, with a carry of 1.

Assume that the previous decimal 33 (binary 100001) is added to decimal 77 (binary 1001101):

$$\begin{array}{r} 1001101 \\ +\ \ 100001 \\ \hline 1101110 \end{array} = \text{decimal } 110$$

There is a special rule for adding columns of binary numbers. If the number of the ones in any single column is greater than 2, divide the number of ones by 2. The number of times that 2 will divide into the number of ones is the amount of ones carried to the next column. If the number of ones divides evenly by 2, then the column total is written as 0. If the column does not divide out evenly, but has a remainder of 1, then the column total is written as 1.

For example:

$$\begin{array}{r} 101 \\ 101 \\ 101 \\ \hline 1111 \end{array} = \text{decimal } 15.$$

In the first (right-hand) column there are three ones. This is more than 2, so it is divided by 2. Two goes into 3 once, with a remainder of 1. Therefore, the remainder of 1 is written for the first-position (right-hand) digit, and the 1 is carried into the second position. This results in:

```
  1
 101
 101
 101
-----
   1
```

In the second column, there is 1 plus 0, plus 0, plus 0, or simply 1. This is less than 2, so the special rule does not apply. Instead, 1 + 0 = 1, and the second-position (middle) total is 1. This results in:

```
 101
 101
 101
-----
  11
```

In the third (left-hand) column, there are three 1s. This is more than 2, so it is divided by 2. Two goes into 3 once, with a remainder of 1. Therefore, the remainder of 1 is written for the third position, and the 1 is carried into the fourth position. This results in:

```
1
  101
  101
  101
 -----
  111
```

In the fourth position, there is a 1 (carried over) plus blanks or zeros. This is less than 2, so the special rule does not apply. Instead, 1 + 0 = 1, and the fourth-position total is 1. This results in:

```
  101
  101
  101
 -----
 1111 = decimal 15.
```

1–2.3 Subtracting Binary Numbers

The rules for subtracting binary numbers are:

$$\begin{aligned}0 - 0 &= 0\\ 1 - 1 &= 0\\ 1 - 0 &= 1\\ 0 - 1 &= 1, \text{ with a borrow of 1.}\end{aligned}$$

Assume that the previous decimal 33 (binary 100001) is subtracted from decimal 77 (binary 1001101):

```
 1001101
 -100001
 -------
  101100 = decimal 44.
```

1–2.4 Multiplying Binary Numbers

The rules for multiplying binary numbers are:

$$\begin{aligned}1 \times 1 &= 1\\ 0 \times 1 &= 0\\ 1 \times 0 &= 0\\ 0 \times 0 &= 0\end{aligned}$$

Assume that the decimal 9 (binary 1001) is multiplied by decimal 3 (binary 11).

```
  1001
  × 11
  ----
  1001
 1001
 -----
 11011 = decimal 27.
```

Note that multiplication is a form of adding and shifting. In logic circuits, there are many systems for multiplication, but they can be classified into two groups. In one group, multiplication is done by adding (with an adder, Chapters 2 and 5) and shifting (with a register, Chapter 4). The other method involves repeated addition. For example, 3 × 9 is, in effect, 9 added together three times (or 3 added together nine times). Multiplication of binary numbers is discussed further in Chapter 6.

1-2.5 Dividing Binary Numbers

In binary, division is a form of subtraction. For example, if decimal 12 is divided by decimal 3, then you can subtract 3 from 12, counting how many times it is subtracted until there is nothing left, or until the remainder is less than 3. Dividing 3 (binary 11) into 12 (binary 1100) results in:

```
     100  or decimal 4
  11√1100
     11
   -----
      00
```

Dividing 5 (binary 101) into 12 (binary 1100) results in:

```
      10  or decimal 2
  101√1100
      101
     -----
      10  or decimal 2 remainder.
```

In logic circuits, division is usually done with subtractors (Chapter 2) and shift registers (Chapter 4). However, binary division can be done with logic subtraction circuits (operating at high speeds) and simple storage registers.

1-2.6 Binary Fractions

To express fractions by binary numbers, proceed initially as in decimals. For example,

$$\frac{3}{5} = \frac{11}{101}$$

This means that the radix uses *negative* powers or exponents. For example, binary 0.0011 (= 0.375 decimal) represents:

$$(0 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times 2^{-3})$$

$$0 + \frac{1}{4} + \frac{1}{8} = \frac{3}{8} = 0.375$$

Figure 1-3 converts a number of common decimal fractions to binary equivalents.