ELECTRONIC DIGITAL TECHNIQUES

Paul M. Kintner, Ph. D.
Manager,
Digital Systems and Products Development
Industrial Systems Division
Cutler-Hammer, Inc.

McGRAW-HILL BOOK COMPANY

New York *San Francisco* *Toronto* *London* *Sydney*

ELECTRONIC DIGITAL TECHNIQUES

 Library of Congress Catalog Card Number 67-24440

ISBN 07-034750-6

67890 MAMM 754321

ELECTRONIC DIGITAL TECHNIQUES

To my parents.

PREFACE

The phenomena of our time sometimes termed the "Computer Revolution" has been based in part on significant advances in electronic digital techniques. These developments have produced digital elements with impressive speed and reliability; equally impressive has been the development of design methods which make practical the assembly of these elements into the complex systems represented by the large-scale digital computer.

There is a growing awareness that these techniques are also of value in such areas as system control and instrumentation: increasingly, digital devices are displacing analog, or nondigital, control and measuring elements, as well as finding wide application in other fields not primarily concerned with computation. There appears to be little doubt that this trend will continue and even accelerate; present developments in solid-state technology are giving an increased efficiency and reliability for digital elements; most important, there is promise of a significant reduction in the costs of these elements. The result of this will be that an increasing number of engineering designers—of all specialities—will find it of value, perhaps mandatory, to have a working knowledge of electronic digital principles.

This book has been written to fulfill what appears to be a need: an exposition of electronic digital technology, but in a form other than the prevalent works for digital computer designers. This difference is not in reducing the material of the computer design text to a superficial treatment, but is one of emphasis: the nature and applications of counters, pulse generators, rate scalers, analog-digital converters, etc.—of somewhat passing interest to the computer designer—are now of major importance; such subjects as multiplication and division algorithms can be touched briefly.

In addition, such a book calls for a departure from the style of the computer design text. Most readers will not be computer design specialists; many will not even be electrical engineers. The more esoteric terms of the computer design vocabulary are best avoided and hueristic reasoning substituted for rigor in such areas as transient behavior of circuits.

As well as attempting to meet these goals, this book has been planned for self-instruction. In as far as space limitations have permitted, few subjects have been introduced without carrying their development to the point where it is hoped that most readers

can achieve a satisfactory understanding without the help of an instructor. As a contribution towards this, short exercises have been inserted at frequent intervals throughout the text. This is a frank attempt to get the reader involved in the exposition, for much of the text relies on the reader "thinking things out" through the exercises. Answers have been provided in most cases to exert, in a small degree, a guiding and corrective influence on the reader.

The synthesis and simplification of logic functions are developed in Chap. 1. The approach relies heavily on tabular and diagrammetric ideas because it is felt that these are most easily assimilated and remembered, and are quite satisfactory for the nonspecialist.

Chapter 2 covers a subject often of painful concern to the system designer: the translation of the results of logic design into circuits. This chapter also introduces some basic electronic digital elements, the diode and transistor, with a discussion which avoids the subject of solid-state physics. A special attempt has been made to clarify the translation from logic into inverting circuits, a task which seems especially confusing to the beginning designer. Chapter 2 also discusses the microelectronic (integrated circuit) form of the diode-transistor digital circuit as well as the implementation of logic through the logic symbol standard MIL-STD-806-B.

Chapter 3 takes up number systems and codes. Special attention has been given to decimal and reflected codes, often encountered in control and instrumentation.

Chapters 4 and 5 are concerned with applications based on the material of the first three chapters. Chapter 4 considers such applications as decoding, encoding, and number comparisons; Chapter 5 takes up arithmetic operations such as binary and decimal addition.

Chapter 6 begins the development of the basis for sequencing digital systems with a study of the memory element or flipflop. The capacitor-diode gate is taken up at this point because of its widespread use with memory elements and as an introduction to the use of resistance-capacitance discharge circuits in digital devices.

Chapter 7 is concerned with applications of groups of memory elements (registers) to such operations as serial arithmetic, binary-to-decimal, and decimal-to-binary operations.

Chapter 8 discusses a basic system sequential device: the electronic counter. This most useful digital system tool is considered in many ramifications and a variety of applications shown.

Chapter 9 investigates the time-based logic signal or pulse. A basis for the analysis of resistance-capacitance discharges is developed for understanding and designing pulse generators. Applications are given in the area of system sequencing and synchronizing.

Chapter 10 discusses a useful device based on a combination of the counter and the pulse generator: the rate scaler or multiplier, which is finding increasing application in the areas of speed and position control.

Chapter 11 considers the problem of the programming or sequencing of digital systems. This is discussed from a general viewpoint and a modular approach to this type of control given.

Chapter 12 finishes with a discussion of input-output devices involving a translation between external units and the digital system.

I would like to express my thanks to those who have aided in the preparation of this book. Much of the material derives from consulting with members of the Industrial Systems Division and Industrial Electronics Division (AIL) of Cutler-Hammer, Inc. Dr. Rex Hubbard and Mr. Jack Linsley have my special thanks for reviewing much of the material. The permission of the management of Cutler-Hammer, Inc. to write this book and their encouragement is gratefully acknowledged.

Mrs. Audrey Rogers and Miss Laurel Belding performed the stenographic work; I am in debt to them for their patience and efficiency.

This book could only be written by encroaching—over several years—on time normally used for family activities. This indirect contribution of my wife and children, made without complaint, is gratefully acknowledged.

PAUL M. KINTNER

CONTENTS

1

THE SYNTHESIS OF LOGIC OPERATIONS

In this chapter we begin our study of electronic digital techniques by considering the synthesis of digital switching operations, usually referred to as logic design. The basic concepts of the logic signal, the logic list, and the fundamental logic operations will be presented.

1.1. Introduction and Definitions

The behavior of a digital control operation usually finds its first expression in a verbal statement: "The motor shall operate if and only if both push-buttons are pushed"; "The motor shall operate if either or both push-buttons are pushed"; "The motor shall be controlled independently from two different points." Still more complicated: "Add A + B."

The task of the designer is to synthesize a system from elementary digital devices which will carry out the operation called for by the verbal statement. This chapter is devoted to the development of design methods which lead to such a synthesis in a systematic orderly manner. It is true that, for simple problems, an intuitive, or "common sense" approach will suffice for obtaining at least an adequate solution. But for most significant problems a systematic approach is of great assistance to the designer in obtaining an efficient design.

The electronic digital devices used for the synthesis of logic operations carry such names as diodes and transistor switching circuits; each can respond to, and generate, two-valued signals: values termed "closed," "open," "negative," "positive." Although a design method might be based on a particular device, an efficient procedure should be general; this means that we must transform our problem into a general language based on symbols, just as we do when we use arithmetic and algebra.

Accordingly, we define an abstract digital control signal, one only indirectly related to hardware. Based on common usage, it will be termed a *logic signal*.*

Definition: The logic signal will be designated by the letters A, B, C where the signal is an input, and by the letters X, Y, Z when an output. The two values of the signal will be designated by the symbols "1" and "0".

It should be emphasized that the "1", "0" as used here do *not* imply numeric properties. Their choice is a convenience which will be apparent later. We will find that when digital and logic signals do represent numbers—and this is basic in a digital computer—the relationship is specified in the form of *codes*, a subject which will be considered in a later chapter.

We can now rephrase our original verbal statement of digital behavior, for instance, the statement: "The motor shall operate if and only if both push-buttons are pushed" can be restated

Z = "1", if *and* only if A and B = "1"

Implied in this restatement are relationships between logic and physical values. For example:

Z = Motor
"1" = Motor "operating"
"0" = Motor "not-operating"
A, B = Push-buttons
"1" = Push-buttons "pushed"
"0" = Push-button "not pushed"

Now the assignment of "1", "0" to the control values of the motor and push-buttons was arbitrary but significantly affects the resulting logic statement. For example, if "1" were assigned to motor "not-operating," the resulting statement would be

Z = "1" if either A or B = "0"

It is important that, although the logic statements appear to be different, they lead to identical behavior when translated into physical switching circuits; it is only necessary to be consistent in considering the relationship between the physical signals and logic signals. This relationship is important enough to warrant a special definition:

---

*"Logic" as used here and in other areas of digital design does not indicate a superior intellectual basis but derives from the fact that the mathematics used was originally developed for the analysis of philosophical reasoning.

Definition: The *logic assignment* is the value of a physical switching signal, "open," "closed"; "positive," "negative" associated with the "1" of its logic signal representation.

Exercise: A plant foreman and safety engineer each describe the action of a two-button control of a punch-press. The foreman states: "The press will operate only if both buttons are pushed"; the engineer: "The press is safe if either button is open." Translate these into logic statements making the appropriate logic assignments.

A more difficult control problem: "The motor shall be controlled independently from two switches," or somewhat more explicitly: "The state of the motor shall change—on to off or off to on—whenever the state of one of the switches changes; the state of the motor shall not change if the state of both switches changes simultaneously." This behavior can be stated in a much more precise way through a table where we systematically list all of the input logic possibilities (there will be a finite number of such possibilities) and then state for each input value a corresponding output value. Such a table will be termed a *logic list*.

Definition: A *logic list* consists of a table giving the output logic values for a set of input logic values.

Let us first describe the mechanics of constructing a logic list; we will then define the motor control problem in terms of a list.

The simplest logic list is one based on a single input logic signal. The input portion of this list is

$$\begin{array}{cc} A & 01 \end{array}$$

where A designates the input signal and the two possible logic values have been listed horizontally from left to right as shown. The list is completed by stating what the output is to be for each column of the list. For example, an obvious output list is

$$\begin{array}{cc} A & 01 \\ \hline Z & 01 \end{array}$$

This would be the listing, for example, for the single point control of a motor where logic assignments of "Motor-on" = "1" and "Button-on" = "1" have been made.

Exercise: There are four possible lists for the case of a single input. One is given above (Z = 01); another possible is Z = 11 which describes a situation where the output is constant and independent of the input. What are the other two output lists?

The list can be expanded to two inputs (A, B) through the following convention: repeat the single-input list for B = "0" and again for B = "1". To illustrate,

A 01 01
B 00 11

The problem postulated above: the two-point control of a motor can be stated in logic form through a two-input list. The output portion is based on the following: the motor logic value is to change whenever the logic value of a switch changes; the motor value is not to change if *both* switches change in value. The resultant list is

A 0101
B 0011
Z 0110

It is readily verified that the logic criteria is satisfied for all possible ways of moving through the input combinations.

The logic list is readily expanded to three-inputs (A, B, C) by repeating the two-input list for C = "0" and again for C = "1":

A 0101 0101
B 0011 0011
C 0000 1111

The general rule for adding a new variable N + 1 to a list is apparent: Repeat the list for N-variables once for N + 1 = "0" and again for N + 1 = "1". Also the list can be written row-by-row by observing the systematic pattern for successive rows: the first is 010101 . . . ; the second: 00110011 . . . ; the third: 0000111100001111 . . .; etc.

Exercise: Write a four-input list with an output satisfying the following: the output is "1" if there is an odd-number of "1s" in the input.

Ans. Z = 0110 1001 1001 0110.

1.2. The Fundamental Logic Operations

Our aim in synthesis is to interconnect a group of elementary digital devices which will carry out the desired system operation: the logic counterpart is to group a set of elementary logic operations which will carry out the desired system logic operation, in our case, the logic operation specified by a logic list. There are a number of basic logic operations suitable for this; the set corresponding to the English words of AND, OR, and NOT are the easiest

to visualize and most natural to use. They will be termed the *fundamental logic operations*.

The NOT-operation. The NOT-operation is a logic operation on a single logic signal defined by the following list:

$$\begin{array}{ll} A & 01 \\ Z & \overline{10} \end{array}$$

In words: the NOT-operation gives an output logic value which is the opposite of the input logic value. The operation is shown symbolically by placing a bar over the input symbol, viz., $\overline{A}$. Further, the list just given could be written $Z = \overline{A}$. Given an output list Z, $\overline{Z}$ is written by taking opposite logic values.

Example: Given Z = 0001 1111 find $\overline{Z}$. Writing opposite values, there results $\overline{Z}$ = 1110 0000.

Exercise: Prove that $\overline{\overline{A}} = A$ by "computing" first $\overline{A}$ (from A = 01) and then $\overline{\overline{A}}$.

The NOT-operation, performed on a logic signal, is termed *inverting* or *complementing* the signal. The NOT-operation can be represented graphically as shown in Fig. 1-1, where the "logic flow" is from left-to-right. There are a number of graphic symbols in common use for the showing of the NOT-operation (and the other fundamental operations); we will write the operation type in a box as shown to avoid the possibility of misinterpretation.

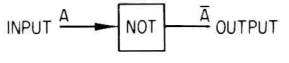

Fig. 1-1. Graphical representation of the NOT-operation.

The AND-operation. The AND-operation is defined as follows for two-inputs:

$$\begin{array}{ll} A & 0101 \\ B & \underline{0011} \\ Z & 0001 \end{array}$$

and for three-inputs,

$$\begin{array}{lll} A & 0101 & 0101 \\ B & 0011 & 0011 \\ C & \underline{0000} & \underline{1111} \\ Z & 0000 & 0001 \end{array}$$

The extension to N-inputs is evident. In words: "The output has the logic value of "1" if and only if all inputs have the value "1". The symbolic representation of the operation is Z = A.B or

Z = A.B.C. A "computing table" can be written based on the list definition; for two inputs,

A B		A. B
0. 0	=	0
0. 1	=	0
1. 0	=	0
1. 1	=	1

We see that the table is the same as a multiplication table for the numbers 1 and 0. This somewhat artificial result is due entirely to the choice of symbols, "1" and "0", for representing logic signal values; but it is convenient to have the operation placed into a familiar form: we then carry out an AND-operation on a logic listing simply by multiplying out the positions.

Example: We have two output lists: P = 0101 0111 and Q = 0001 0101; what is P. Q? The two lists are arranged one below the other and multiplied position-by-position:

P	0101 0111
Q	0001 0101
P. Q	0001 0101

Exercise: Prove that A.A = A and A.$\overline{A}$ = "0" by computing out the operation, starting with A = 01.

The parallelism between the AND-operation and multiplication is the basis for speaking of A.B as the *logic product* of A, B. The graphical representation (for use in this book) is shown in Fig. 1-2.

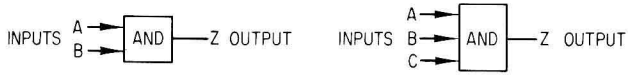

Fig. 1-2. Graphical representation of the AND-operation.

The OR-operation. The OR-operation will be defined by the following lists:

A	0101
B	0011
Z	0111

A	0101	0101
B	0011	0011
C	0000	1111
Z	0111	1111

with the extension of the definition to N-inputs evident. The symbol for the OR-operation is Z = A + B, Z = A + B + C. This use of addition symbols comes from the following logic computing table:

A	B	Z
0	0	0
1	0	1
0	1	1
1	1	1

where arithmetic addition holds except for the last row—an unfortunate failure in our attempt to parallel arithmetic operations by choosing the symbols "1" and "0" for logic values. The graphical symbol is as shown in Fig. 1-3. The operation is sometimes termed a *logic sum* and a list value can be readily computed by using the computing table given above.

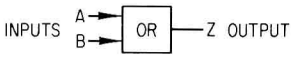

Fig. 1-3. Graphical representation of the OR-operation.

Example: Given P = 0101 0111 and Q = 0001 0101, compute P + Q.

P	0101	0111
Q	0001	0101
P+Q	0101	0111

Combined operations. The fundamental logic operations are usually used in combinations and in cascade. For example: Z = A.B + C (the result of an AND-operation on A, B is an input to an OR-operation along with C). The graphical symbol is as shown in Fig. 1-3. The operation is sometimes termed a *logic diagram*, is shown in Fig. 1-4. The list value of the cascaded operations is readily computed:

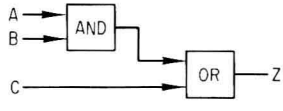

Fig. 1-4. The logic diagram of Z = A.B + C.

A	0101	0101
B	0011	0011
C	0000	1111
A.B	0001	0001
+C	0000	1111
Z	0001	1111

Z = A.B + C

Exercise: Compute the list value of $Z = (A+\overline{B}).(\overline{A}+C).(B+\overline{C})$.

Ans. Z = 1000 0001.

Dual operations. The reader is undoubtedly familiar with duality in such fields as circuit analysis; duality exists between logic operations, which is not surprising considering the two-valued nature of the signals upon which they are based.

Definition: The dual of any logic operation is obtained by inverting all values in the logic list, both input and output, which defines the operation.

It is readily shown that the OR-operation is a dual of the AND-operation:

A	0101
B	0011
A.B	0001
$\overline{A}$	1010
$\overline{B}$	1100
$\overline{A}+\overline{B}$	1110

Exercise: Show that the AND-operation is a dual of the OR-operation. Does the NOT-operation have a dual?

The duality between the AND and OR-operations indicates that we do not really need both for synthesis: given one, we can devise the equivalent of the other by performing NOT-operations on the inputs and output of the first. However, all three (and their physical counterparts) are customarily used to avoid inverting and to give a more easily visualized logic operation.

1.3. The Minterm and Maxterm

We have discussed the computation of the logic value (in the form of a listing) for a given logic expression consisting of cascaded fundamental logic operations; our aim in synthesis is the reverse: given a logic value, how do we devise cascaded logic operations to generate that value? The basis of our method will be the synthesizing of two special logic outputs.

The minterm. The minterm is a logic operation with an output list having a single "1". For example,

A	0101	0101
B	0011	0011
C	0000	1111
m_2	0010	0000

is a minterm based on three-inputs, where m_2 indicates a minterm with its "1" in position-2 (based on numbering from left-to-right