Symbolic Logic, Boolean Algebra and the Design of Digital Systems

By the Technical Staff of COMPUTER CONTROL COMPANY, INC.

FOREWORD

We sincerely hope that our many readers will find the contents of this manual helpful in obtaining an understanding of the fundamental principles of Symbolic Logic and the application of these principles to the design of digital systems.

COMPUTER CONTROL COMPANY wishes to express its gratitude to the members of its technical staff for their many contributions which were compiled and combined into the manuscript of which this publication is the end result.

21559 1 15

TABLE OF CONTENTS

SYMBOLIC LOGIC

PRACTICAL APPLICATIONS

ABOUT 3C

Symbolic Logic

INTRODUCTION

This publication represents an attempt to present in a clear and understandable form the fundamentals of Symbolic Logic as applied to the logical design of digital systems. COMPUTER CONTROL COMPANY, Inc., utilizes the techniques of Symbolic Logic continuously in the design of its digital products and systems. It is the intention of this section to make this knowledge available in a single coherent presentation so that others in our industry may benefit similarly. We hope the readers will endeavor to let us know whether or not we have succeeded.

HISTORICAL

The first clear notion of a system of mathematical logic was formulated by the German mathematician, Leibniz. The first concrete contributions to the achievement of a systematized logic were made by Augustus de Morgan (1806-1876) and George Boole (1815-1864). The entire later evolution may be said to stem from Boole's monumental contribution titled, *An Investigation of the Laws of Thought on which are Founded the Mathematical Theories of Logic and Probabilities.* This was published in 1854 but was overlooked as an interesting academic novelty for a great many years. Recognition as a mathematical masterpiece finally came from Whitehead and Russell in their *Principia Mathematica* (1910-1913). In 1928 a classic text on the subject titled, *Mathematical Logic* by Hilbert and Ackerman was published in German and later in English.

The object of Symbolic Logic was to provide an instrument of exact, analytical, constructive thought; i.e., to effect a means of avoiding the pitfalls of semantics in language. As a subject in a university curriculum, it is presented by the philosophy department quite as often as by the mathematics department because of its involvement with language and semantics. However, like all other discoveries in pure science, it was only a matter of time for an expanding technology to find a practical engineering use for this new knowledge. The date which marks the beginning of this new period is generally recognized to be 1938 when Dr. Claude E. Shannon of the Bell Telephone Laboratories published his paper titled, *A Symbolic Analysis of Relay and Switching Circuits.* The past decade has seen the widespread application of digital computers, data-processing systems, automatic control systems, and other types of digital equipments towards the needs of scientific and industrial progress. The design and development of these modern systems of tremendous complexity have caused Symbolic Logic to become a vital area of knowledge to the digital systems designer. He is always concerned with the minimization or simplification of the amount of equipment required to perform a given function. Symbolic Logic provides the means for rigorously attacking this problem. There is more to be said on the subject of minimization and a later section will return to this most important topic.

NUMBER SYSTEMS

An important adjunct to an understanding of the fundamentals of Symbolic Logic is a knowledge of the binary number system. This will be reviewed only briefly here in view of the prevalence of knowledge on this topic among digital engineers and systems designers.

In some prehistoric age, early man learned to count by becoming aware of a one-to-one correlation between his possessions of one sort or another and his fingers. Conceivably this may have resulted from an attempt to communicate to his neighbors quantitative information for bartering purposes. In any event the concepts of magnitude and quantity were developed when man learned to count with his fingers. And because he had ten fingers the decimal number system has been in use ever since.

Our method of writing a number using Arabic numeral symbols to indicate a quantity or magnitude is actually a shorthand.

for example:

796 is shorthand for $7 \times 100 + 9 \times 10 + 6 \times 1$, which may also be written as $7 \times 10^2 + 9 \times 10^1 + 6 \times 10^0$.

It is obvious that any decimal number can be expressed in this manner. On further reflection it can be seen that the ascending powers of 10 represent a sophisticated solution to early man's problem of keeping track of the number of ten counts he had made on his fingers when he became involved with large quantities. The modern Theory of Numbers gives the term "radix" to this base 10 of our decimal number system and calls the digits 0 through 9 "coefficients". It also shows that number systems can be devised using any other base or radix. In general, the number of coefficients required is equal to the value of the radix if the number system is to be capable of expressing all quantities in a non-redundant manner.

BINARY NUMBER SYSTEM

The binary number system uses the radix 2. It has two coefficients, therefore, 0 and 1. It is the number system which might have been adopted if our early ancestors possessed one finger on each hand instead of five. Any quantity or magnitude which can be expressed decimally can also be expressed in binary form.

for example:

17 decimal $= 1 \times 10^1 + 7 \times 10^0$
17 written in binary = 10001

which is shorthand for . . .

$1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$ which adds up to 17 as we commonly know it.

It can be seen that a sort of transformation has been achieved. Any quantity or magnitude can be represented equally well by either system or by all other conceivable systems for that matter. But the decimal system requires knowledge or memory of ten different coefficients while the binary system involves only two. However, it requires many more of these two binary coefficients in appropriate combination to represent a given magnitude than it does to represent the same magnitude using decimal coefficients. The human mind has no difficulty in memorizing and learning to manipulate the ten coefficients 0 to 9 and so the decimal system works out well and saves us considerable inconvenience as compared to systems with a smaller base. However in electrical or electronic equipment using relays, vacuum tubes, magnetic cores, transistors, etc., it is both difficult and uneconomical to design and manufacture circuits which have ten discrete, stable, and easily controllable states due to tolerance problems, state of the component art, etc. Therefore the majority of our digital machines do not operate directly in the same decimal number system that we ourselves prefer. It is not difficult and it is economically feasible to design bistable circuits which are efficient and reliable. Accordingly most of our modern computers perform their internal arithmetic operations using the binary number system. However even if this were not the situation; as for example in the case of digital equipments which do not perform arithmetic operations; a knowledge of the binary number system and binary arithmetic provides a valuable aid to the understanding and appreciation of the capabilities of Symbolic Logic. This is so because our two-valued binary number system coefficients possess a direct correspondence to the two-valued or dyadic form of the basic elements of Symbolic Logic which will be discussed in later sections.

NOTE: The simplest number system is the unary system (radix = 1, coefficient = 1) which we are all familiar with and sometimes use to keep tallies . . .

for example:

1 = 1, 2 = 11, 3 = 111, 4 = 1111, 5 = ~~1111~~, 6 = ~~1111~~ 1, etc.

BOOLEAN ALGEBRA

Symbolic Logic is often referred to as Boolean algebra in commemoration of its prime originator. It is significant also that it is called an algebra and not a geometry or an arithmetic or anything else. The significance lies in the structural resemblance which it bears to the subject of algebra. This resemblance perhaps makes a comparison between the two worthwhile for orientation purposes. A further purpose of the following brief comparison is that it illustrates the relative simplicity of Symbolic Logic as compared to algebra.

ALGEBRA

1. Algebraic Variables
   Letter symbols are used to represent dependent and independent variables whose numerical values can range from plus infinity to minus infinity; whose number form can be real, imaginary, complex, rational, irrational, integral, fractional, etc.

2. Fundamental Operations
   addition $(+)$

$c = a + b$

c	a	b
2	1	1
3	1	2
4	2	2
9	5	4

etc. in infinite variety.

subtraction $(-)$

$c = a - b$

c	a	b
1	2	1
2	3	1
−2	1	3
5	9	4

etc. in infinite variety.

Similarly for multiplication and division which, however, can also be derived from the more basic operations of addition and subtraction.

3. Commutative and Associative Laws
   $a + b = b + a$
   $a - b = -b + a$
   $a + (b + c) = (a + b) + c$
   $a(bc) = (ab)c$

SYMBOLIC LOGIC

1. Binary variables
   Letter symbols are used to represent dependent or independent variables which are *always* simply two-valued (dyadic, binary). They may be expressed as one, zero; true, false; pulse, no pulse; plus voltage, minus voltage; etc.

2. Fundamental Operations
   conjunction (symbol $\cdot$)

$c = a \cdot b$
c = a and b
c is true only when 'a' and 'b' are both true simultaneously.
True = 1, False = 0

no other possibilities exist.

c	a	b
0	0	0
0	0	1
0	1	0
1	1	1

disjunction (symbol $\vee$)

$c = a \vee b$
c = a or b
c is true whenever either 'a' or 'b' are true.
T = 1, F = 0

c	a	b
0	0	0
1	0	1
1	1	0
1	1	1

negation

$a = \overline{b}$
'a' is true if 'b' is false; 'a' is false if 'b' is true; 'a' is the negation of 'b'

a	b
0	1
1	0

Other types of logical operations exist and are found useful by the non-engineer logician. However these may always be derived from the three above which are most readily implementable by electronic means. Consequently, the digital engineer is generally concerned only with the above operations of conjunction, disjunction, and negation.

3. Commutative and Associative Laws
   $a \cdot b = b \cdot a$
   $a \vee b = b \vee a$
   $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
   $a \vee (b \vee c) = (a \vee b) \vee c$

Figure 1 Comparison of Algebra vs Symbolic Logic

BINARY VARIABLES

As stated in the previous comparison table, a binary variable can vary between two and only two values. It corresponds directly to the "bit" of information theory. The term "bit" is synonymous with binary variable and gained popular usage in the digital engineering field. The two values of a binary variable are commonly represented as true, false; one, zero; plus voltage, minus voltage; pulse, absence of pulse; open relay contact, closed relay contact; etc.

COMBINATIONS OF BINARY VARIABLES

Since the binary variable can have only two possible states, a finite number of binary variables taken together must yield a finite number of possible combinations. For example if our binary variable is represented by the position of a toggle switch, two switches (representing two variables) can result in only four possible combinations. These are ON-ON, ON-OFF, OFF-ON, and OFF-OFF. Three switches would give eight possible combinations; four would give sixteen; etc. In general the number of possible combinations can be seen to be equal to 2^n where n equals the number of switches; i.e., binary variables.

FUNCTIONS OF BINARY VARIABLES

The concept of a function of binary variables is somewhat more difficult to grasp and therefore also, to explain. It might be stated simply as a particular grouping or combination of combinations of binary variables. To carry our switch-position analogy further, let us assume three binary variables represented by the three switches: A, B, and C. There are eight possible configurations of ON and OFF which could be set up on A, B, and C. Representing ON by the numeral 1 and OFF by the numeral 0, these may be tabulated as in Fig. 2.

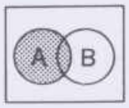
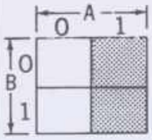
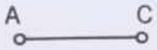
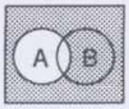
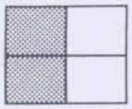
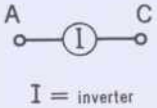
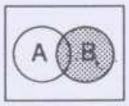
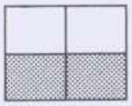
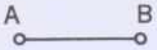
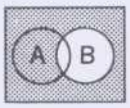
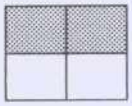
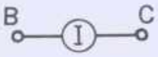
Some arbitrary action could be predicated solely on the occurrence of any one of the eight possible combinations of A, B, and C. The predicated action could also be based on the occurrence of more than one combination. It could be initiated by either combination 1 or combination 5; or by any of combinations 3 or 4 or 7; or any combination except 8; etc. These combinations of combinations represent functions. There are a finite number of possible functions of n variables. The reader should satisfy himself that in general the number of combinations $= 2^n$ where $n =$ the number of variables and that the number of functions $= 2^{2^n}$.

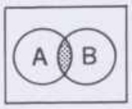
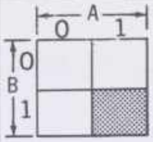
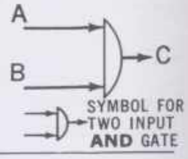
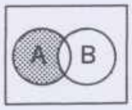
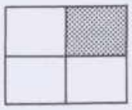
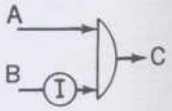
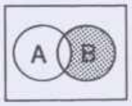
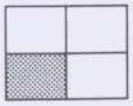
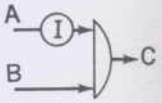
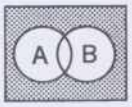
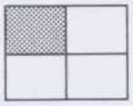
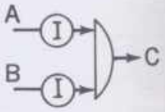
Combination	A	B	C
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1

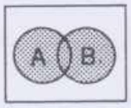
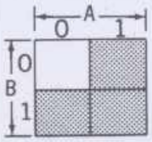
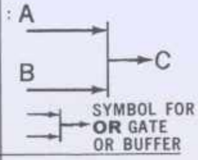
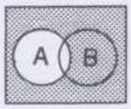
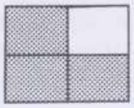
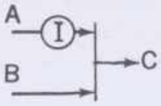
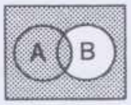
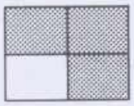
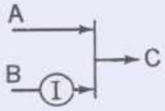
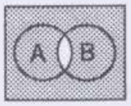
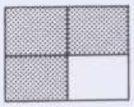
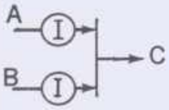
Figure 2 Combinations of Three Binary Variables

GRAPHICAL METHODS OF PRESENTATION

In keeping with the truism that a picture is worth a very large number of words, the tendency has been to devise a diagrammatic approach to the presentation of combinations and functions of binary variables. The first such diagrams were created by a 19th century mathematician, John Venn, and are called Venn diagrams. The modern approach comprises a rectangular chart called a "Truth Table" which could be described as a Venn diagram recreated in Cartesian coordinates. Fig. 3 illustrates the complete case for all functions of two variables using logical equations, electronic symbols for implementation, Venn diagrams, and truth tables.

LOGICAL EQUATION C = f(A, B)	VENN DIAGRAM C = f(A, B)	TRUTH TABLE C = f(A, B)	ELECTRONIC IMPLEMENTATION C = f(A, B)
$C = A$			
$C = \bar{A}$			
$C = B$			
$C = \bar{B}$			

LOGICAL EQUATION C = f(A, B)	VENN DIAGRAM C = f(A, B)	TRUTH TABLE C = f(A, B)	ELECTRONIC IMPLEMENTATION C = f(A, B)
$C = A \cdot B$			
$C = A \cdot \bar{B}$			
$C = \bar{A} \cdot B$			
$C = \bar{A} \cdot \bar{B}$			

LOGICAL EQUATION C = f(A, B)	VENN DIAGRAM C = f(A, B)	TRUTH TABLE C = f(A, B)	ELECTRONIC IMPLEMENTATION C = f(A, B)
$C = A \vee B$			
$C = \bar{A} \vee B$			
$C = A \vee \bar{B}$			
$C = \bar{A} \vee \bar{B}$			

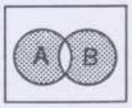
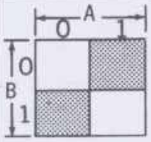
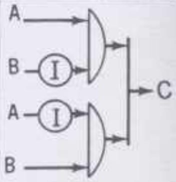
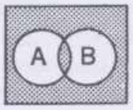
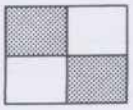
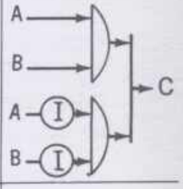
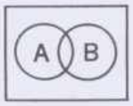
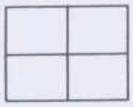
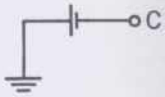
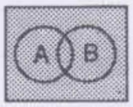
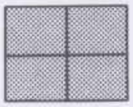
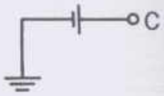
LOGICAL EQUATION C = f(A, B)	VENN DIAGRAM C = f(A, B)	TRUTH TABLE C = f(A, B)	ELECTRONIC IMPLEMENTATION C = f(A, B)
$C = (A \cdot \bar{B}) \vee (\bar{A} \cdot B)$			
$C = (A \cdot B) \vee (\bar{A} \cdot \bar{B})$			
$C = 0$			
$C = 1$			

Figure 3 Functions of Two Binary Variables

GENERAL THEOREM

Any binary function may be expressed as a disjunctive combination of conjunctions. Electronically speaking, this means that the function may be represented by a group of AND gates or conjunctions whose outputs are buffered together into an OR configuration. Conversely, *any function may also be expressed as a conjunctive combination of disjunctions* which is implementable by a group of OR gates (buffers) whose outputs feed into an AND gate. Minimization of any given function from its truth table will lead to either of these two general forms, as desired. If the function is to be implemented by means of germanium diode gating structures, this attribute of the truth table is a very advantageous one since it leads to only two levels of gating structure and since the not quite "ideal" characteristics of the diodes make it mandatory to minimize the number of cascaded gates.

DeMORGAN'S THEOREM

Another useful rule to the logical designer is DeMorgan's Theorem which states that any binary expression is equal to the negation of the expression obtained by changing all conjunctions to disjunctions and vice-versa and by replacing each variable with its negation.

some examples:

$\overline{A \vee B} = \bar{A} \cdot \bar{B}$

$\overline{A \cdot B} = \bar{A} \vee \bar{B}$

$(A \cdot B) \cdot [B \vee \bar{C} \vee (D \cdot E)] = \overline{(\bar{A} \vee \bar{B}) \vee [\bar{B} \cdot C \cdot (\bar{D} \vee \bar{E})]}$

MINIMIZATION

As previously mentioned, the designer of digital equipments is inevitably concerned with the need to obtain the simplest, most direct implementation of his logical functions. Minimization of equipment results in maximization of economy and of reliability, all else being equal. Of the various methods devised to obtain the minimization of binary functions, the truth table or chart method has achieved the most widespread acceptance because of its relative ease of usage and because it provides a visualization of the function that makes its characteristics more readily perceivable. The truth tables for all sixteen functions of two variables have already been listed. Fig. 4 illustrates the layout of the truth tables for functions of three and four variables.

In similar fashion charts for five, six, seven, and even eight variables can be created. Note that a three variable chart has eight squares, a four variable chart has sixteen squares, and that an n variable chart will have 2^n squares. Also note that the tabular listing of the coordinate values of the variables is done in a "reflected" sequence rather than a straight binary sequence such that only one variable at a time changes state between any two adjacent coordinates. This is done to facilitate determination of redundant variables in the conjunctive terms by quick inspection. Its need will become obvious from consideration of the following steps to be pursued in achieving the minimization of binary functions.

Figure 4 Three and Four Variable Truth Tables
(squares are numbered for discussion purposes)

		0	1	C
A	B			
0	0	1	▨ 2	
0	1	▨ 3	▨ 4	
1	1	5	▨ 6	
1	0	7	8	

Three variable chart for
$f = \bar{A} \cdot B \cdot \bar{C} \vee \bar{A} \cdot \bar{B} \cdot C \vee \bar{A} \cdot B \cdot C \vee A \cdot B \cdot C$

		0	1	1	0	C
		0	0	1	1	D
A	B					
0	0	▨ 1	2	▨ 3	4	
0	1	5	6	7	8	
1	1	▨ 9	10	11	▨ 12	
1	0	13	14	▨ 15	16	

Four variable chart for
$f = A \cdot B \cdot \bar{C} \vee \{ [\bar{A} \cdot \bar{B}] \cdot [(\bar{C} \cdot \bar{D}) \vee (C \cdot D)] \} \vee A \cdot \bar{B} \cdot C \cdot D$

Step 1. Create the chart with the appropriate number of squares in accordance with the number of variables involved and with the variable coordinates in reflected sequence.

Step 2. (Algebraic Approach). Shade in the appropriate squares to completely represent the given function. This is perhaps the most difficult step in the procedure. It is straightforward if the function is already in the form of disjunctively connected conjuncts. If it is in the form of conjunctively connected disjuncts, DeMorgan's Theorem can be used to effect the reversal. If the function is not in a neat and symmetrical form, then it can be expanded by means of the commutative, associative, and distributive laws. Also the expansion of the function may be expedited by manipulations based on the following simple relationships:

$A \vee \bar{A}$ = always true
$A \cdot \bar{A}$ = always false
$A \cdot A = A$
$A \vee A = A$
$A \vee (A \cdot B) = A$
$A \cdot (A \vee B) = A$
$A \vee (\bar{A} \cdot B) = A \vee B$
$A \cdot (\bar{A} \vee B) = A \cdot B$

The validity of these minimizations may be established by means of the two variable chart.

In the examples of Fig. 4 the charting of the two functions proceeded as follows:

Three Variable Chart:

$\bar{A} \cdot B \cdot \bar{C}$ shaded square No. 3
$\bar{A} \cdot \bar{B} \cdot C$ shaded square No. 2
$\bar{A} \cdot B \cdot C$ shaded square No. 4
$A \cdot B \cdot C$ shaded square No. 6

Four Variable Chart:

$A \cdot B \cdot \bar{C}$ shaded squares No. 9 and 12,
$[\bar{A} \cdot \bar{B}] \cdot [(\bar{C} \cdot \bar{D}) \vee (C \cdot D)]$ were expanded to:
$(\bar{A} \cdot \bar{B} \cdot \bar{C} \cdot \bar{D}) \vee (\bar{A} \cdot \bar{B} \cdot C \cdot D)$ and shaded squares No. 1 and 3, and $A \cdot \bar{B} \cdot C \cdot D$ shaded square No. 15.

ALTERNATE AND PREFERABLE STEP 2 (Graphical Approach)

Again the requirement of Step 2 is to shade in the appropriate squares to completely represent the given function. The advantage of the graphical approach is that common sense is substituted for knowledge of the logical algebraic manipulations in shading in the functions. Chart manipulation replaces algebraic manipulation in accomplishing the identical results correctly and easily. For example, if a term in the expression to be displayed is $(A \cdot B)$, shade in all squares that have $A = 1$ and $B = 1$ as coordinates. If the term is $A \vee B$, shade in all squares that have $A = 1$ or $B = 1$ as coordinates. Do this on separate charts (or on the same chart in different colors) for all single terms and then in similar fashion combine charts (or colors) to get the final unique table for the function.

Application of step 3 to the chart which has just been derived will show that $f_3 = B \vee (\bar{A} \cdot C)$

Figure 5 Chart Shading for $f = (A \cdot B) \vee (\bar{B} \cdot \bar{C})$ Using Graphical Approach

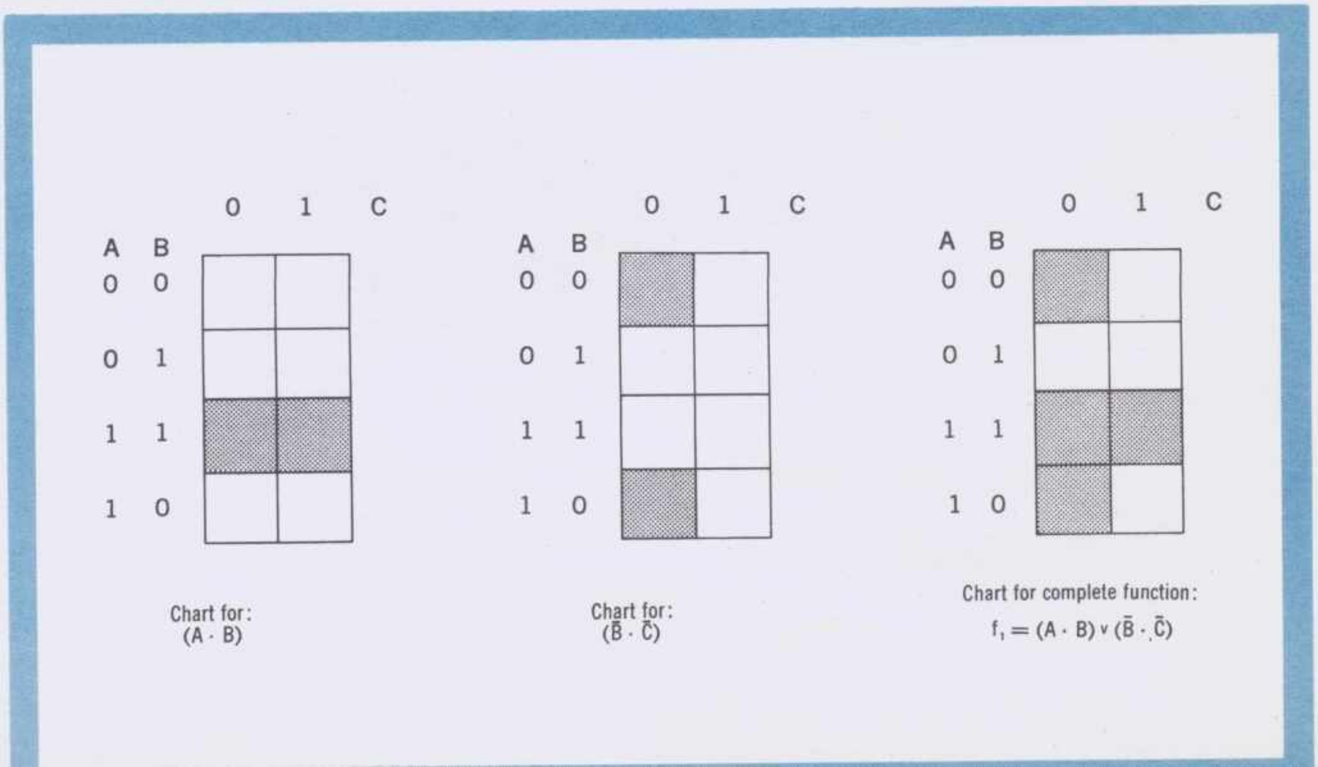


Chart for: $(A \cdot B)$

Chart for: $(\bar{B} \cdot \bar{C})$

Chart for complete function: $f_1 = (A \cdot B) \vee (\bar{B} \cdot \bar{C})$

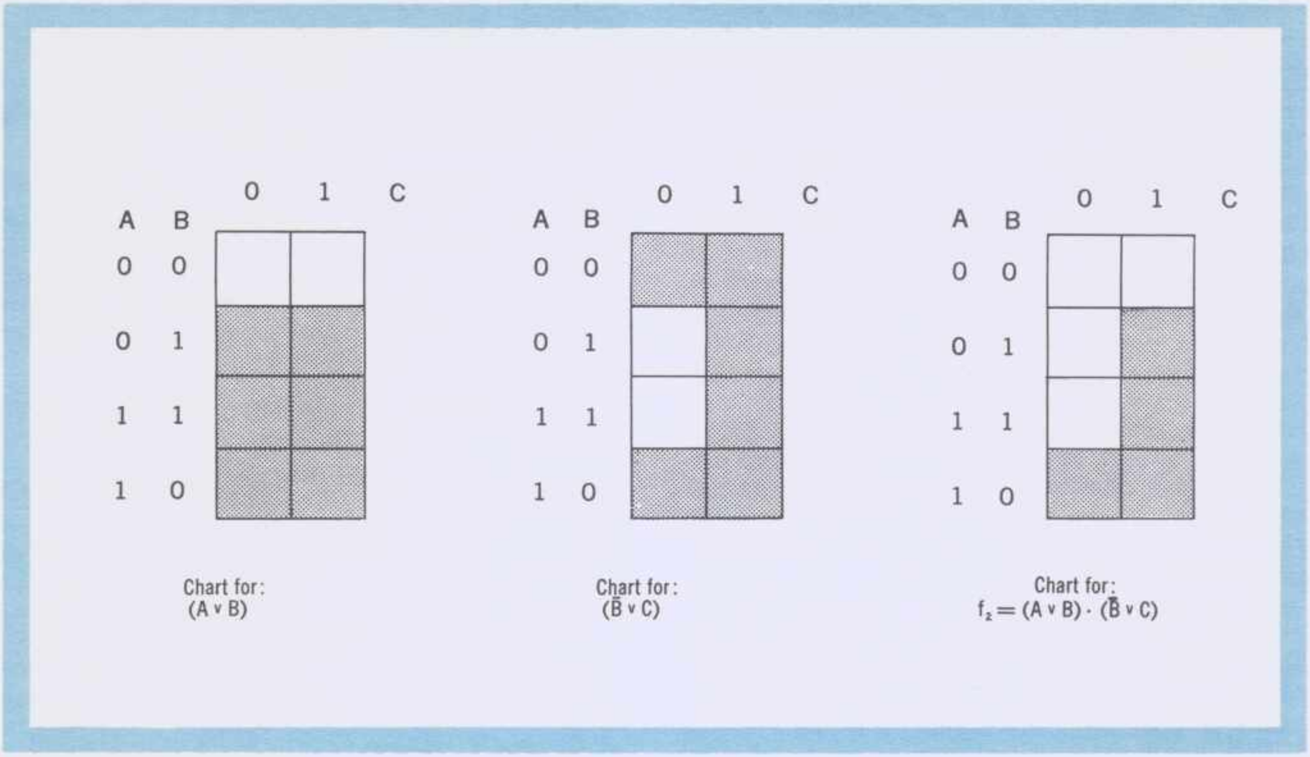


Figure 6 Chart Shading for $f = (A \vee B) \cdot (\bar{B} \vee C)$ Using Graphical Approach

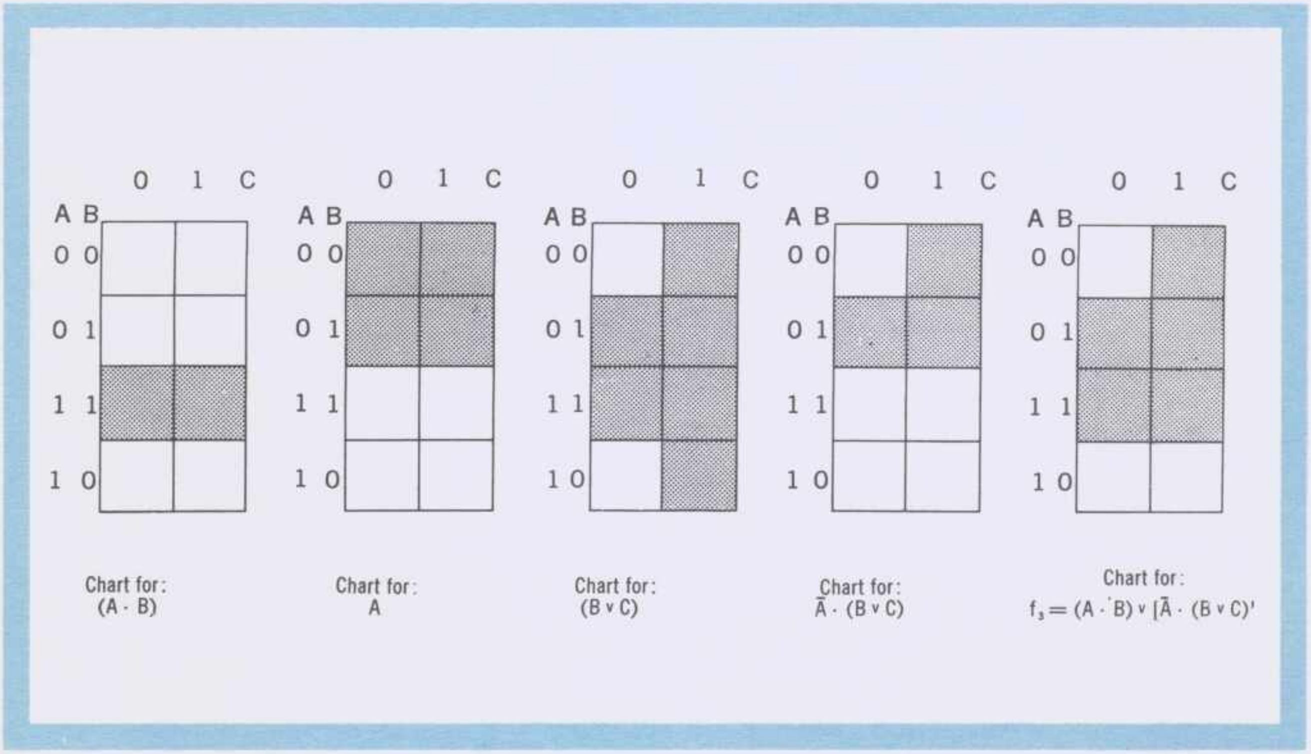


Figure 7 Chart Shading for $f = (A \cdot B) \vee [\bar{A} \cdot (B \vee C)]$ Using Graphical Approach

Step 3 will be described first for functions of four variables but the same general technique applies to all other cases.

The brute force implementation of the function requires an AND gate for each square of the chart and as many inputs to each gate as there are variables in the expression. This form of the function was derived in order to shade the chart squares and does not represent any simplification. The simplification or minimization is achieved by inspecting the chart for shaded large squares or rectangles comprised of the individual shaded squares such that all of the individual squares are shaded and such that the number of individual squares comprising the larger shaded square or rectangle is one, two, four, or eight. *The two outside columns of the chart are considered to be contiguous* for purposes of this inspection. Similarly, *the top and bottom rows of the chart are also considered to be contiguous to each other.*

Item 1 Eight adjacent squares forming a 2 × 4 rectangle can be represented by a "one-leg" gate (i.e. a direct input to the 2nd level buffer).

Item 2 Four adjacent squares forming either a 2 × 2 or a 4 × 1 rectangle can be represented by a "two-leg" gate.

Item 3 Two adjacent squares sharing the same row or column can be represented by a "three-leg" gate.

Item 4 Remaining single squares each require a "four-leg" gate for representation.

The validity of the statements of Items 1-4 above should be clearly established by the reader to his complete satisfaction before proceeding. Inspection of the shaded areas will indicate that the specified number of gate legs conform to those coordinate variables whose values are constant for the entire area under consideration.

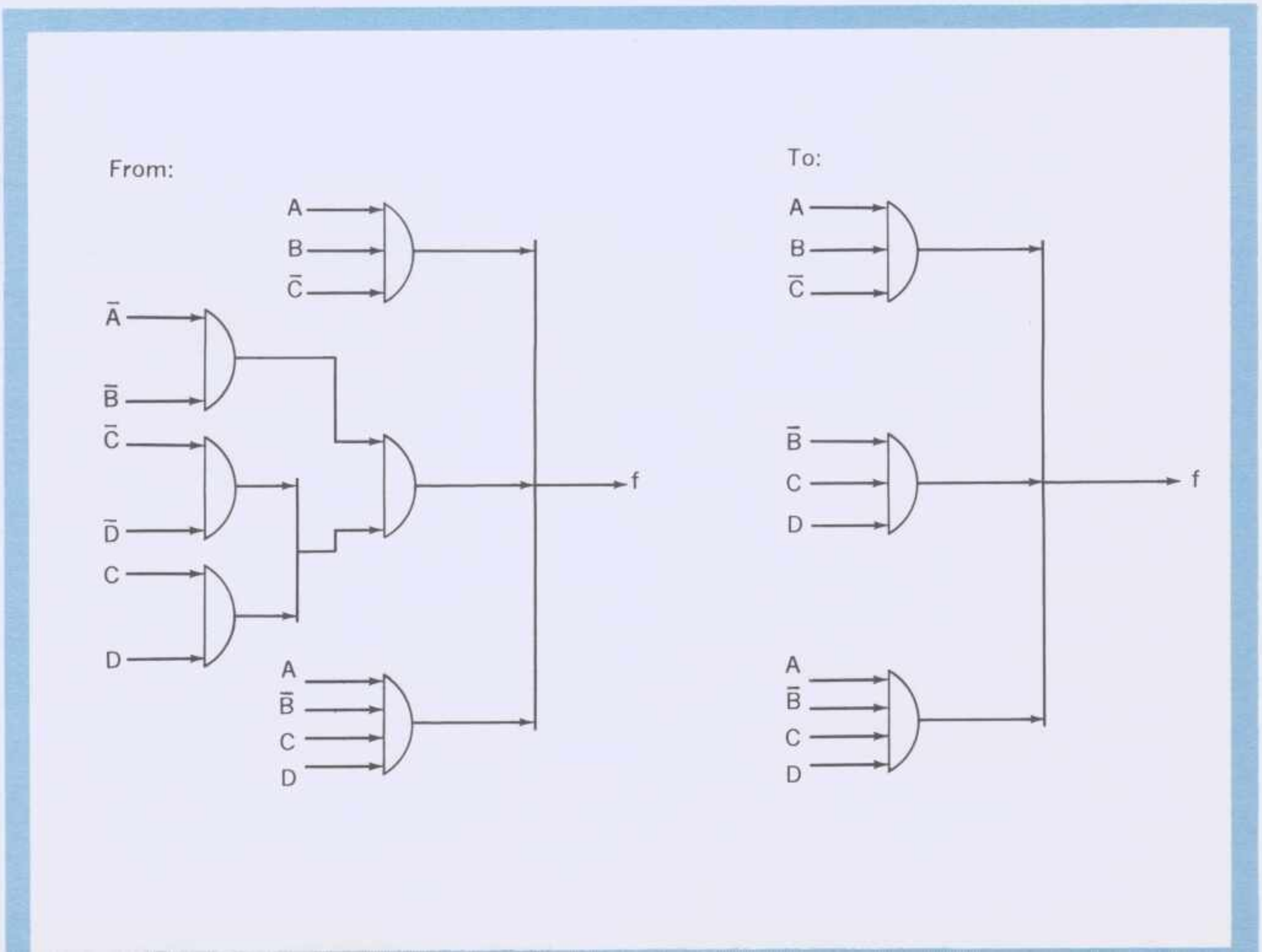
For the specific case of the four variable chart of Fig. 4 it may be seen that:

Square 9 is "adjacent" to Square 12 and yields $A \cdot B \cdot \bar{C}$.

Square 3 is "adjacent" to Square 15 and yields $\bar{B} \cdot C \cdot D$.

Square 1 stands alone and requires $\bar{A} \cdot \bar{B} \cdot \bar{C} \cdot \bar{D}$. therefore, $f = A \cdot B \cdot \bar{C} \vee \}[\bar{A} \cdot \bar{B}] \cdot [(\bar{C} \cdot \bar{D}) \vee (C \cdot D)]\{ \vee A \cdot \bar{B} \cdot C \cdot D$ has been minimized to $f = A \cdot B \cdot \bar{C} \vee \bar{B} \cdot C \cdot D \vee \bar{A} \cdot \bar{B} \cdot \bar{C} \cdot \bar{D}$ and the implementation has been reduced as indicated in Fig. 8.

Figure 8



Step 3 Example for the case of three variables:

The function $f = (\bar{A} \cdot B \cdot \bar{C}) \vee (\bar{A} \cdot \bar{B} \cdot C) \vee (\bar{A} \cdot B \cdot C) \vee (A \cdot B \cdot C)$ of Fig. 4 is already in the desired form for entry in the chart.

$\bar{A} \cdot B \cdot \bar{C}$ shades square No. 3.
$\bar{A} \cdot \bar{B} \cdot C$ shades square No. 2.
$\bar{A} \cdot B \cdot C$ shades square No. 4.
$A \cdot B \cdot C$ shades square No. 6.

There is no 2 × 2 or 4 × 1 shaded area which would require a "one-leg" (direct input) gate. The choice exists between a 2 × 1 area and two separate shaded squares or three 2 × 1 shaded areas which all have square No. 4 in common. Either choice will require three gates but use of the common square No. 4 eliminates one leg on each of the two gates and is therefore preferable.

Therefore:

squares 3 and 4 yield $\bar{A} \cdot B$.
squares 2 and 4 yield $\bar{A} \cdot C$.
squares 6 and 4 yield $B \cdot C$.

and so the function minimizes to:

$$f = (\bar{A} \cdot B) \vee (\bar{A} \cdot C) \vee (B \cdot C)$$

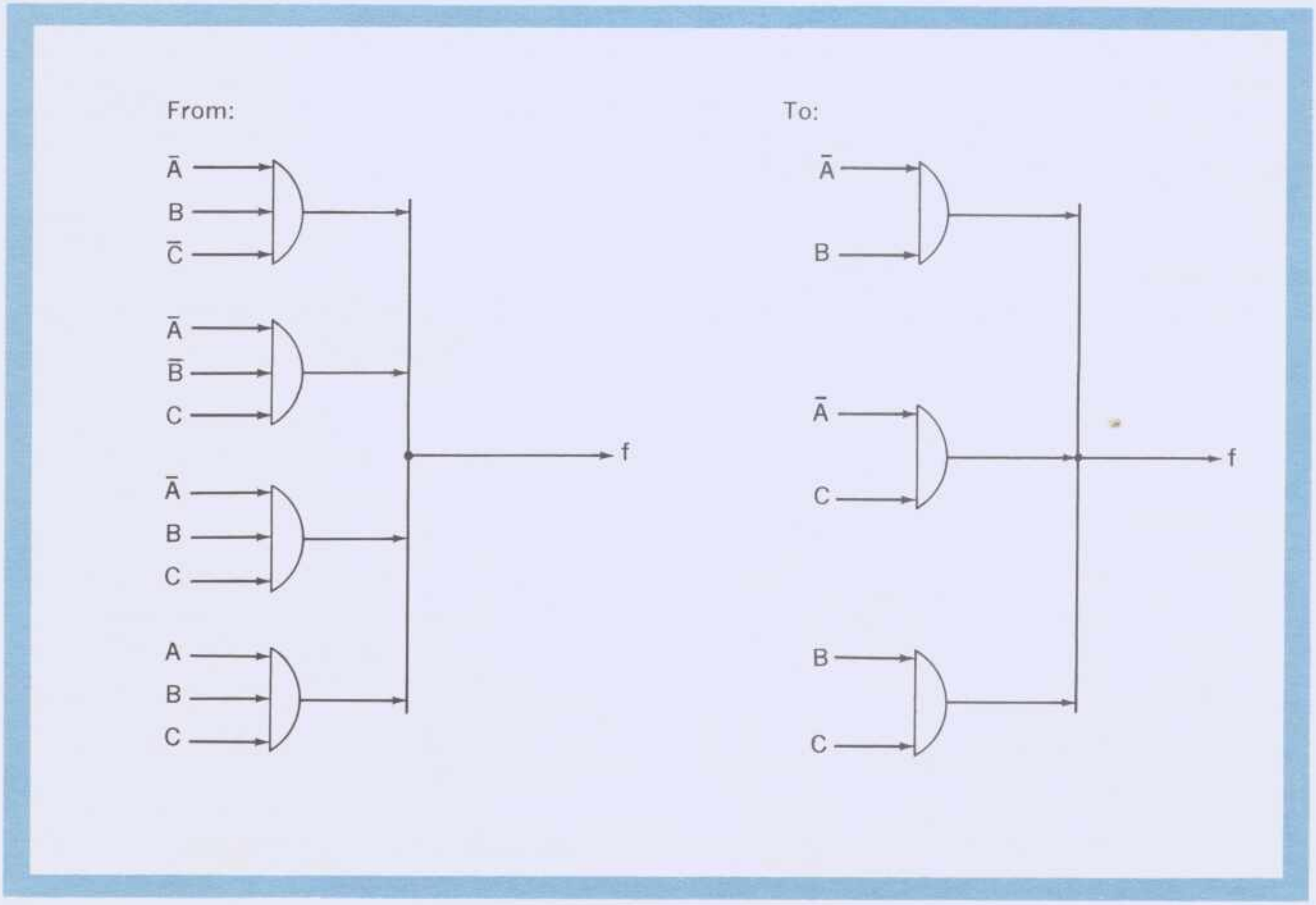
and implementation reduces to the form shown in Fig. 9.

Use of the overlap on Square 4 means physically that the occurrence of inputs $\bar{A} \cdot B \cdot C$ will result in an output from each of the three gates in the minimized implementation, but since the three outputs occur simultaneously and are buffered together, the actual output signal of the logical gating structure is unaffected.

Step 3 for functions of five or more variables:

Five variables are graphed on a three variable by two variable coordinate chart. The situation becomes immediately more difficult although still very manageable. In practice, functions of up to eight or more variables are successfully minimized using the chart method. Functions of higher numbers of variables rarely occur and when they do, can be handled by breaking them up into two or more discrete functions of lesser numbers of variables.

Figure 9



The increased difficulty with five or more variables stems from the fact that a multiplicity of reflected sequences exist for the coordinate values. Consider the case of a three variable reflected coordinate sequence:

A	B	C	
0	0	0	
0	0	1	
0	1	1	
0	1	0	
1	1	0	also reflective
1	1	1	etc.
1	0	1	also reflective
1	0	0	

This means that consideration of adjacent shaded squares is inadequate. Other shaded squares might exist which also could have been adjacent depending on the sequence selected but are prevented from being so by limitations stemming from the two-dimensional aspects of the chart. Careful inspection of the chart will still yield the proper results nevertheless. Also reference to the literature on the subject will disclose the existence of various aids and techniques to assist in making the minimization a routine procedure. These should be consulted when complete familiarity has been achieved for the cases of two, three, and four variables.

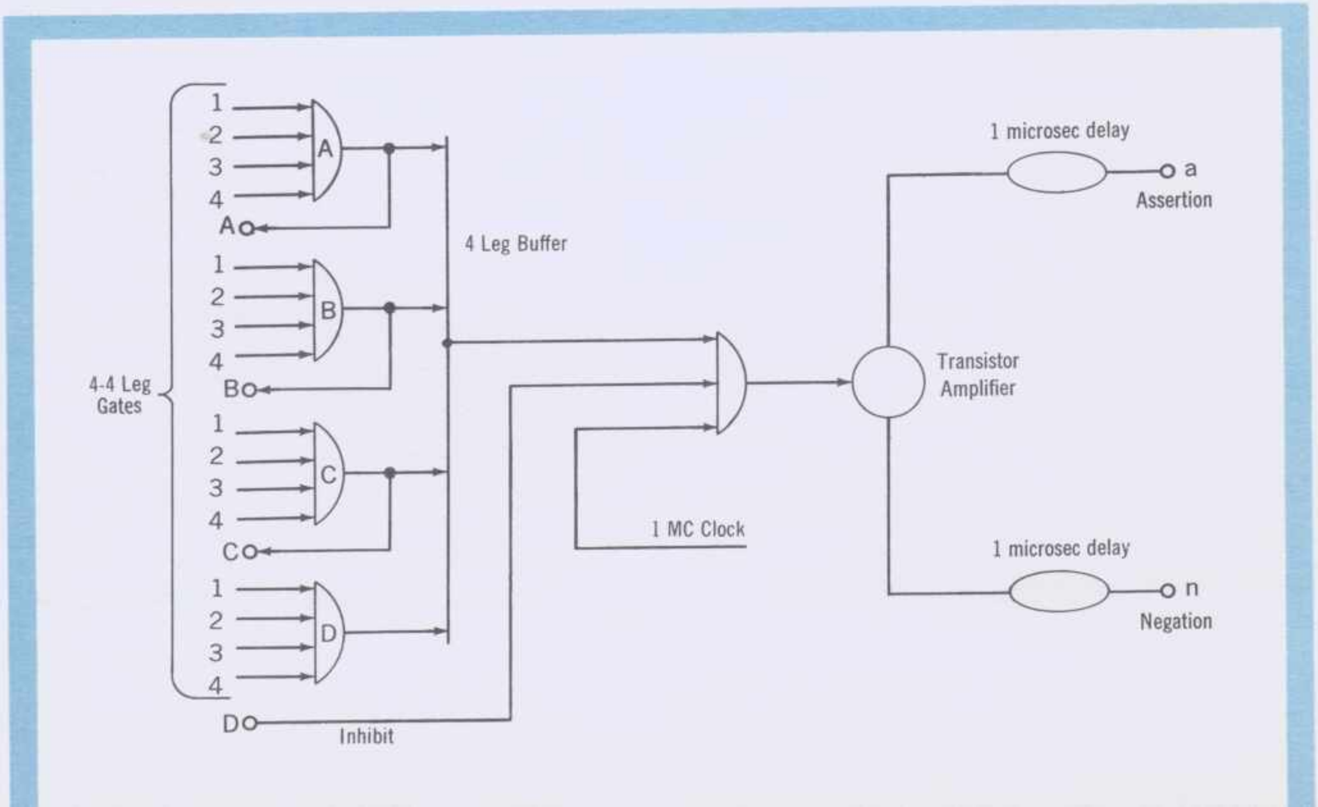
Practical Applications

The following usage examples are based upon the Series T family of 3C-PACs (or T-PACs) which are described in detail in COMPUTER CONTROL COMPANY's Catalog T. (*The reader is strongly urged to familiarize himself with the operation of COMPUTER CONTROL COMPANY's T-PACs by referring to Catalog T as an aid in understanding the examples contained in the following sections.*)

The two level gating configuration of the Model LE-10 T-PAC is the result of careful consideration of the principles which have been described in the preceding sections. It was chosen as being the optimum configuration for the implementation of three, four, and five variable binary functions and as being most permissive to the direct minimization of these functions.

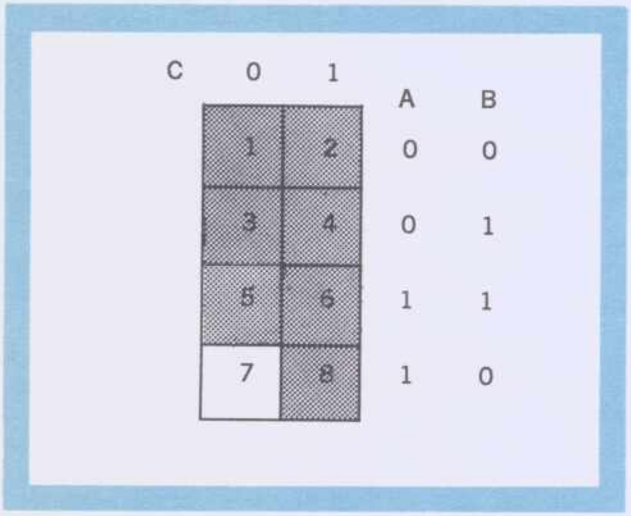
T-BLOC®
3C-PAC®

Figure 10 Logical Representation of T-PAC LOGICAL ELEMENT, Model LE-10



IMPLEMENTING THE NEGATION OF A FUNCTION

Because the LE-10 T-PAC has assertion and negation outputs either the shaded or the unshaded squares of the truth table may be implemented in accordance with whichever requires the least number of inputs. If the unshaded squares (the negation of the function) are implemented then the negation output of the T-PAC provides the required function. Consider the function described by the following truth table:



C	0	1	A	B
	▨ 1	▨ 2	0	0
	▨ 3	▨ 4	0	1
	▨ 5	▨ 6	1	1
	7	▨ 8	1	0

Figure 11

Writing the three variable term for each shaded square (sometimes called the disjunctive normal form) the function is:

$$f = \overbrace{\bar{A} \cdot \bar{B} \cdot \bar{C}}^{1} \vee \overbrace{\bar{A} \cdot \bar{B} \cdot C}^{2} \vee \overbrace{\bar{A} \cdot B \cdot \bar{C}}^{3} \vee$$

$$\overbrace{\bar{A} \cdot B \cdot C}^{4} \vee \overbrace{A \cdot B \cdot \bar{C}}^{5} \vee \overbrace{A \cdot B \cdot C}^{6} \vee \overbrace{A \cdot \bar{B} \cdot C}^{8}$$

in minimized form:

$f = \bar{A} \vee B \vee C$

$\bar{A}$ from squares 1, 2, 3, 4

B from squares 3, 4, 5, 6

C from squares 2, 4, 6, 8

NOTE:
Number above bracket indicates square on truth table of Fig. 11

however consider square 7:

$$f_7 = A \cdot \bar{B} \cdot \bar{C}$$

$$\bar{f}_7 = \overline{A \cdot \bar{B} \cdot \bar{C}} = \bar{A} \vee B \vee C = f, \text{ by}$$

DeMorgan's Theorem

Therefore square 7 could have been implemented directly with the proviso that the negation output of the package now must be used to obtain the function.

FLIP-FLOP

The flip-flop, shown in Fig. 12, is a familiar bistable element. The simplest form of a flip-flop is one that is turned ON by applying a pulse S (set pulse) and turned OFF by applying a pulse R (reset pulse); i.e., once the flip-flop has been turned on by the set pulse, it should remain on (true) until it is turned off by the reset pulse. Conversely, it should remain off (false) until it is turned on again by a set pulse.

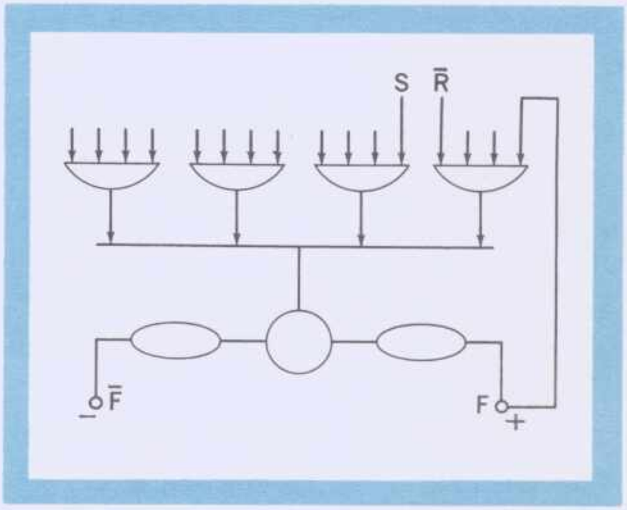


Figure 12 Basic Flip-flop Connections in a T-PAC

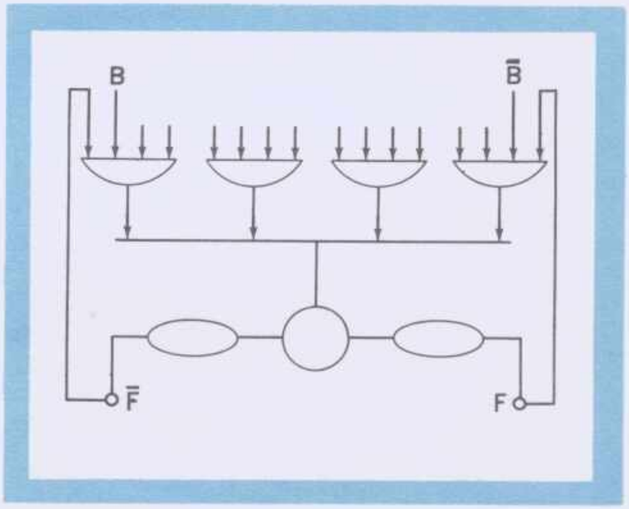


Figure 13 Binary Flip-flop

If symbolic logic is employed, the expression for this device is $S \vee (\bar{R} \cdot F)$, where S is the set pulse, F is the plus output of the flip-flop, R is the reset, and $\bar{R}$ is used to mean "no reset pulse". Hence, the statement $F = S \vee (\bar{R} \cdot F)$ says that this device shall have a true output after it receives a set pulse and shall continue to have a true output until a reset pulse is received. If the statement is examined, it can be seen that the S pulse creates the true pulse F and that the term $(\bar{R} \cdot F)$ maintains this true state. Whenever an R pulse is received, the latter term becomes false, causing the entire statement to be false, hence turning off the device. Thus, a flip-flop has been created. The flip-flop connections used on a T-PAC are shown in Fig. 12.

BINARY FLIP-FLOP

A binary flip-flop, shown in Fig. 13, is a stable device that changes state each time an input pulse B is applied. The symbolic logic statement for this is $F' = (B \cdot \bar{F}) \vee (\bar{B} \cdot F)$. The symbol F', read "F delayed one pulse period", distinguishes the output of a T-PAC from its inputs which occur one microsecond earlier. The characteristic of a binary flip-flop is that if it is off (F' false and $\bar{F}'$ true), a pulse B will turn it on (make F' true). The flip-flop will remain on ($\bar{B} \cdot F$ true) until another pulse B is received. Conversely, if the device is on (F' true and $\bar{F}'$ false), a pulse B will turn it off until another pulse B is received.

SET-RESET AND BINARY FLIP-FLOP

An extension of the flip-flop and binary flip-flop, shown in Fig. 14, is a bistable device that is turned on by a set pulse, turned off by a reset pulse, and has its state changed by the application of the B pulse. The symbolic logic statement for this is:

$$F = S \vee (B \cdot \bar{F}) \vee (\bar{B} \cdot \bar{R} \cdot F)$$

The appropriate connections are shown in Fig. 14.

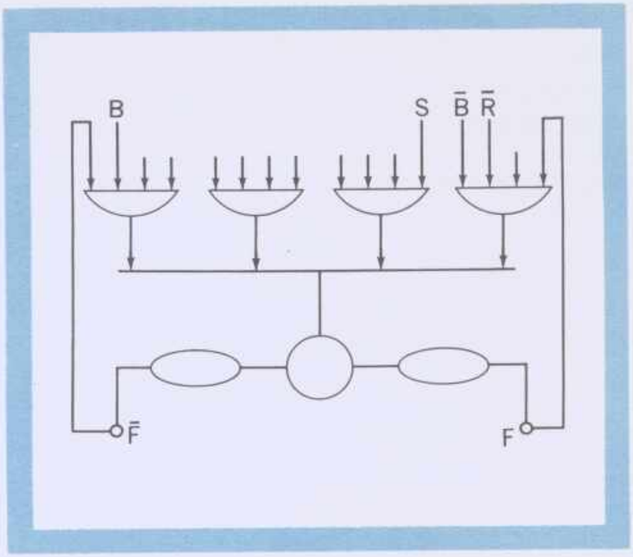
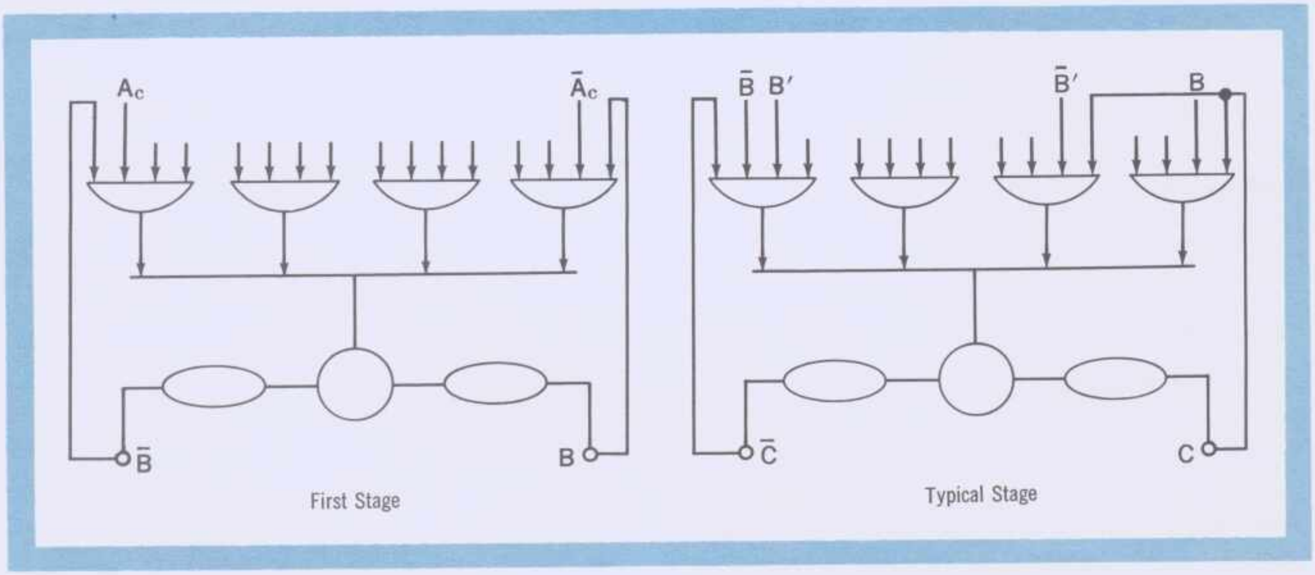


Figure 14 Set, Reset, and Binary Flip-flop

BINARY COUNTER

A binary counter, see Fig. 15, is an n-stage device that will accept count pulses and display the binary version of the accumulated count. The total capacity of this counter is $(2^n - 1)$.

Figure 15 Binary Counter



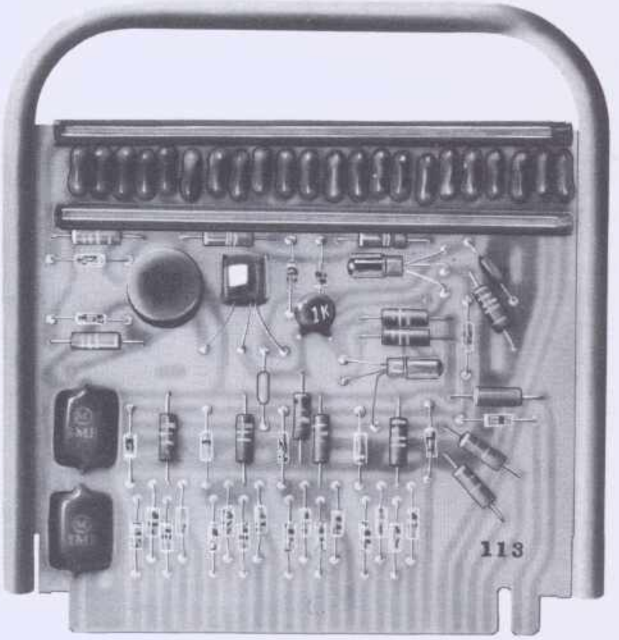
The first stage is a conventional binary flip-flop. Each of the other stages sense for a change of state (1 to 0) of the preceding (low order) stage to determine whether the higher-ordered stage should change state. For example, a two-stage counter can count from 0 to 3, where the binary representations of its contents are:

	C	B
0 =	0	0
1 =	0	1
2 =	1	0
3 =	1	1

The change of state in stage B that should serve to change the state of stage C is the transition of B from 1 to 0. This transition is recognized by sensing the present output of stage B and its output 1 pulse period earlier. When these two outputs are 0 and 1, respectively, stage C should change state. If the count pulse is labelled A_c, then the statements for stage B and a typical stage C are as follows:

$$B = (A_c \cdot \overline{B}) \vee (\overline{A}_c \cdot B)$$
$$C = (\overline{C} \cdot B' \cdot \overline{B}) \vee (C \cdot \overline{B}') \vee (C \cdot B)$$

The last statement shows that if C is false ($\overline{C}$ is true) and B changes from a 1 to a 0 (B' and $\overline{B}$ is true), the first quantity ($\overline{C} \cdot B' \cdot \overline{B}$) is true, causing C to change state and become true. Similarly; if C is true and we have B' and $\overline{B}$ true, then the circulation of C is disabled and C changes state and becomes false. The T-PAC connections for the binary counter are shown in Fig. 15.



T-PAC LOGICAL ELEMENT, Model LE-10

FORWARD-BACKWARD COUNTER

A forward-backward counter, shown in Fig. 16, is a binary counter whose contents are increased by 1 each time an add pulse A is applied, and decreased by 1 each time a subtract pulse S is applied. Each stage of the counter consists of two packages: a flip-flop to store a 0 or a 1 for the stage, and a package to produce a carry or borrow pulse to the next stage. The storage flip-flop in the first stage (low-order stage) must change state upon receipt of either an add pulse or a subtract pulse. The storage flip-flop in any of the remaining stages must change state upon the receipt of a carry or borrow pulse from the preceding stage.

When adding, a carry pulse must be produced from a stage when it is switched from 1 to 0. When subtracting, a borrow pulse must be produced from a stage when it is switched from 0 to 1. The carry and borrow pulses are derived in a single package and delivered to the next stage as one signal.

A typical stage in the counter (other than the first) changes state each time a carry or a borrow signal is received from the preceding stage. At the time a carry or a borrow signal is received by a particular stage, the carry-borrow package associated with that stage determines whether a carry or a borrow signal should, in turn, be transmitted to the next higher-order counter stage. This carry or borrow decision is based on the states of the storage flip-flops of the particular stage and its preceding stage.

If the preceding stage is reset (0) at the time

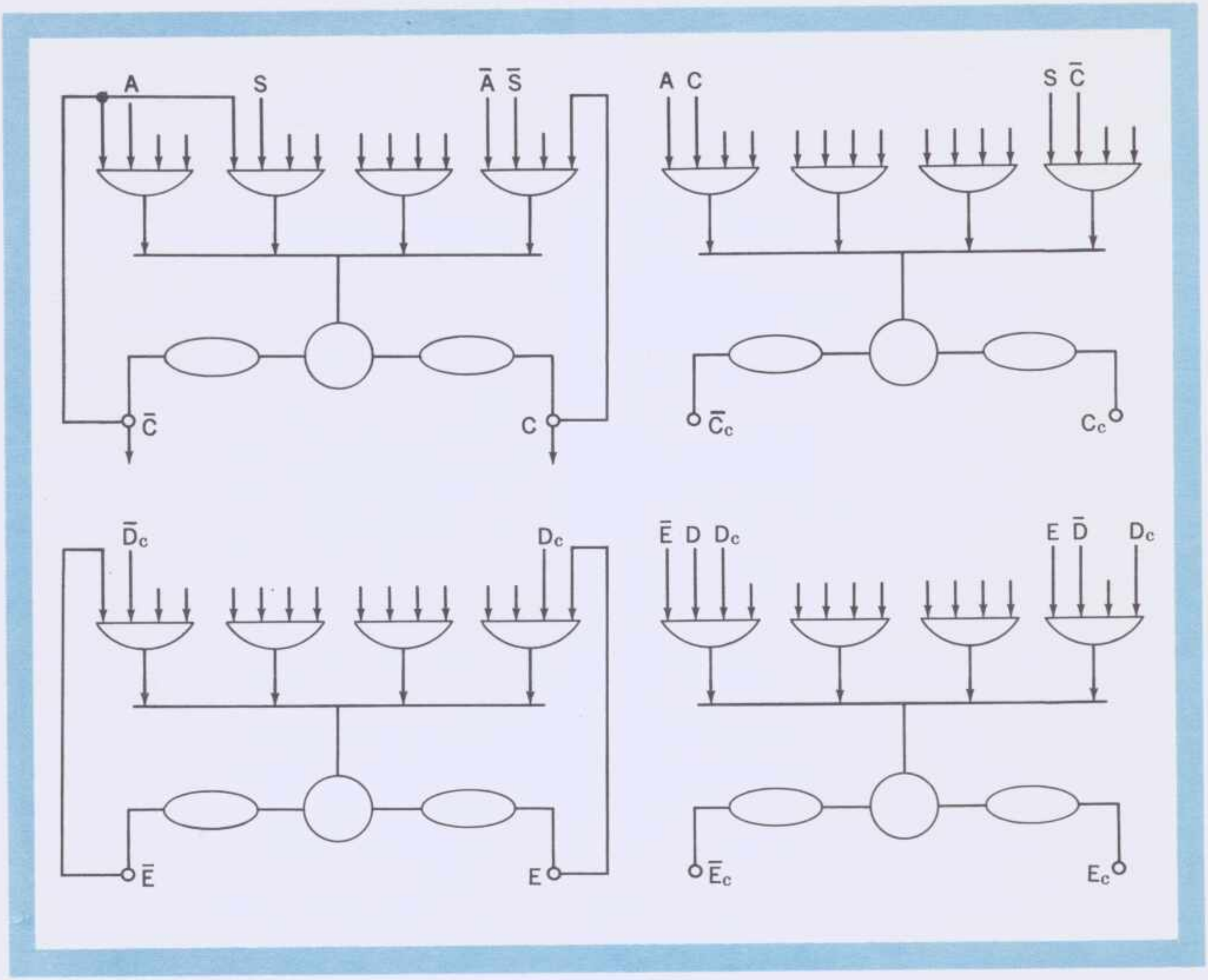


Figure 16 Forward-Backward Counter

that this stage receives a carry or borrow pulse from the preceding stage, then this carry or borrow pulse is a carry pulse. If this stage is set (1) at the time this carry pulse is received, the storage flip-flop associated with this stage should be reset to 0 and a carry pulse transmitted on the carry-borrow line to the next higher-order counter stage. If this stage is reset (0) at the time this carry pulse is received, then the storage flip-flop associated with this stage is set and no carry pulse is transmitted to the next stage.

If the preceding stage is set (1) at the time that this stage receives a carry or borrow pulse from the preceding stage, then this carry or borrow pulse is a borrow pulse. If this stage is reset (0) at the time that this borrow pulse is received, the storage flip-flop associated with this stage should be set to 1 and a borrow pulse transmitted to the carry-borrow package of the next higher-order counter stage. If this stage is set (1) at the time that this borrow pulse is recieved, then the storage flip-flop associated with this stage is reset and no borrow pulse is transmitted to the next stage.

The T-PAC connections for the forward-back-ward counter are shown in Fig. 16 where:

A	=	Add pulse
S	=	Subtract pulse
C	=	First stage storage
C_c	=	First stage carry or borrow pulse
E	=	Typical stage storage
E_c	=	Typical stage carry or borrow pulse

and D and D_c are the storage and carry or borrow pulse of the stage preceding stage E.

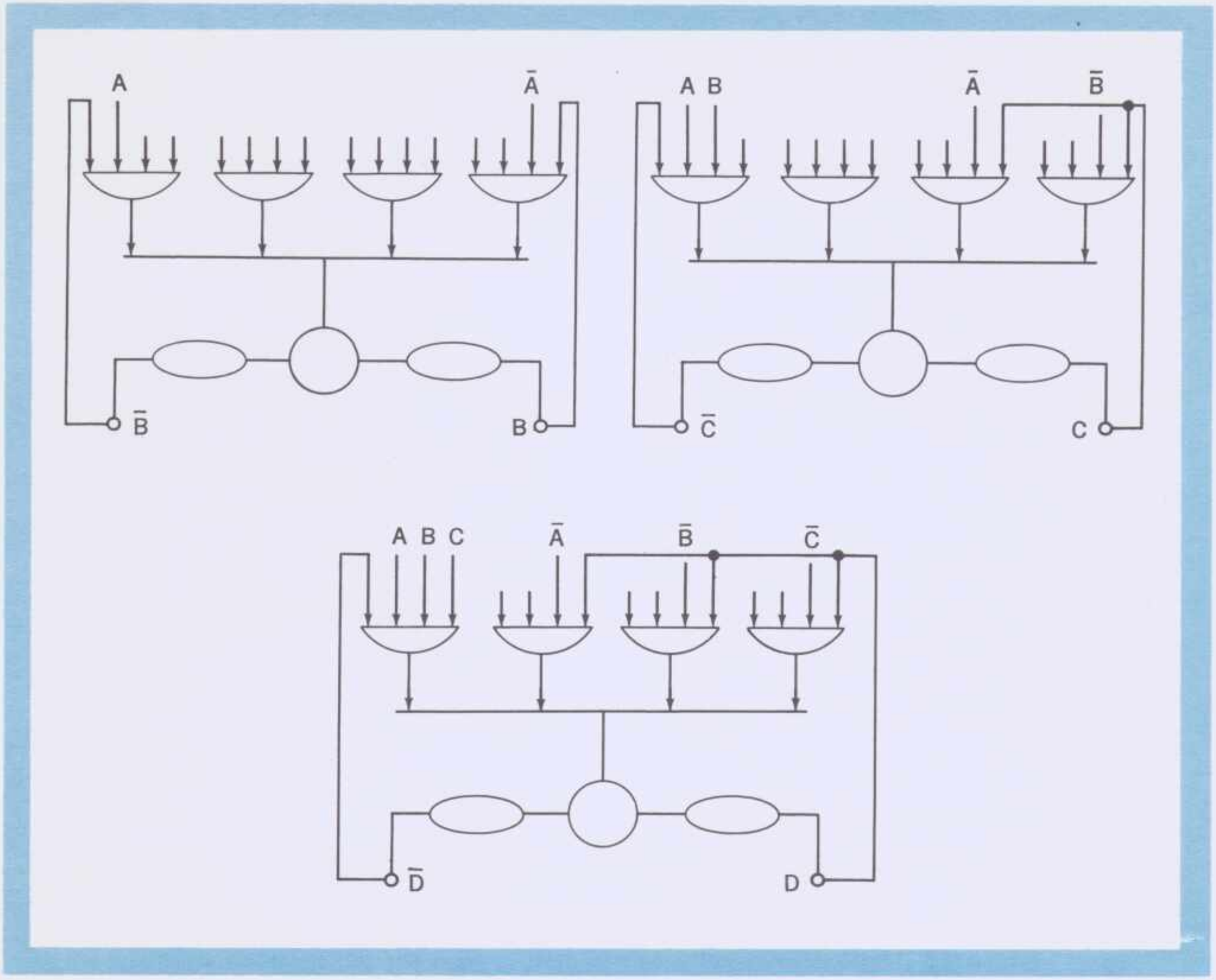


Figure 17 Three Stage Counter

SPECIALIZED BINARY COUNTER

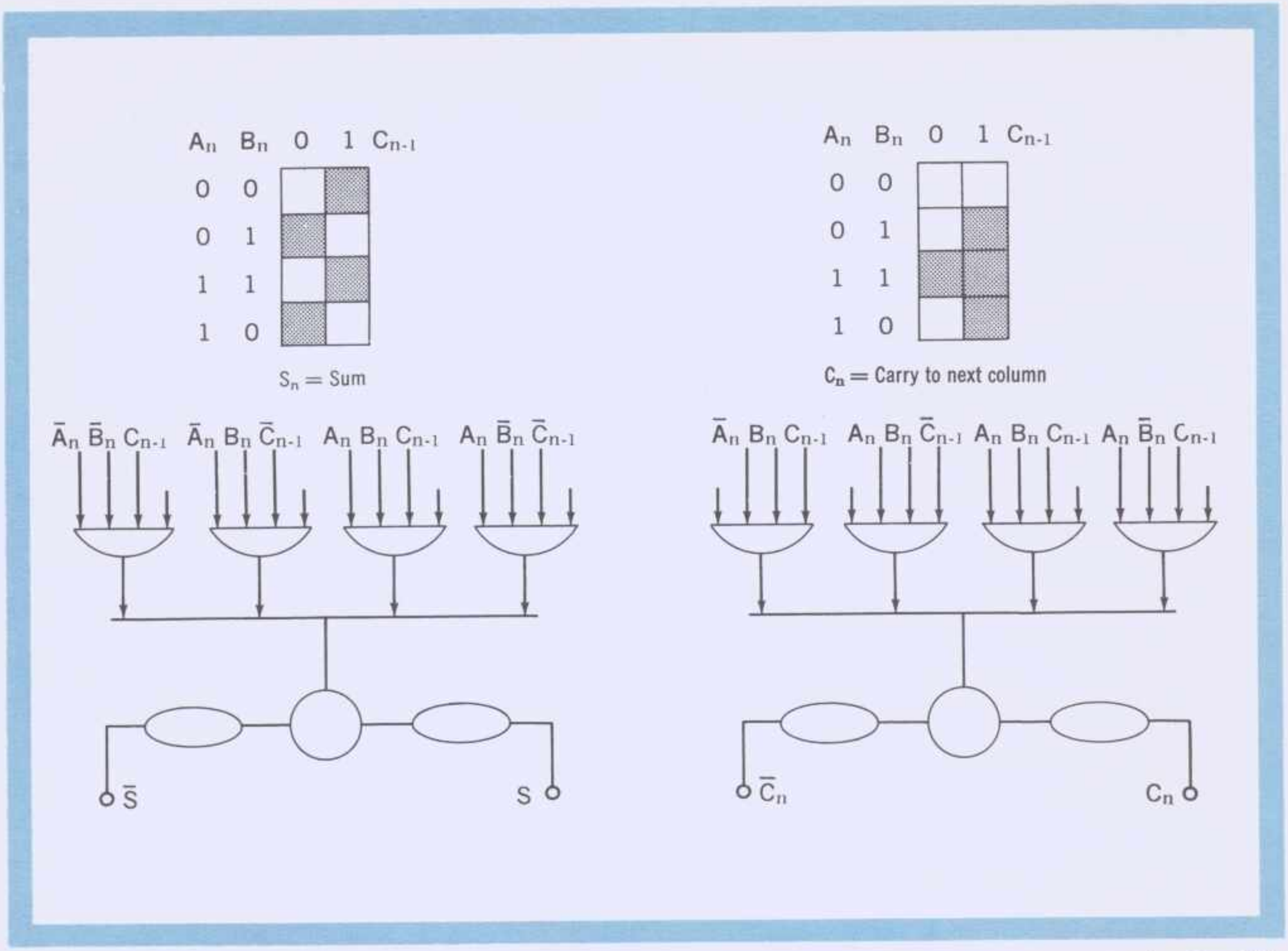
This particular binary counter, shown in Fig. 17, has all stages responding simultaneously when its contents are increased by 1 upon the application of an input pulse A. This type of counter will provide the new accumulated count 1 microsecond after the count pulse has been received. Each stage is a binary flip-flop. The first stage (stage B) changes state every time the count pulse is received. The second stage (stage C) changes state every time stage B is on and a count pulse is received. The third stage (stage D) changes state every time stage B and stage C are on and a count pulse is received. In general, stage n changes state whenever all the preceding stages are on and a count pulse is received. The eight consecutive states of a three-stage binary counter verify this reasoning.

0 = 000	5 = 101
1 = 001	6 = 110
2 = 010	7 = 111
3 = 011	0 = 000
4 = 100	

Letting,

A = The input count pulse
B = The first (low order) stage
C = The second stage
D = The third stage

the T-PAC connections for a three-stage counter of this type are shown in Fig. 17. While a three-stage counter of this type requires only three packages, it should be noted that, for speed, succeeding stages require more than one package each.



A_n B_n 0 1 C_{n-1}
0 0
0 1
1 1
1 0
S_n = Sum
$\bar{A}_n \bar{B}_n C_{n-1}$ $\bar{A}_n B_n \bar{C}_{n-1}$ $A_n B_n C_{n-1}$ $A_n \bar{B}_n \bar{C}_{n-1}$
$\bar{S}$
S
A_n B_n 0 1 C_{n-1}
0 0
0 1
1 1
1 0
C_n = Carry to next column
$\bar{A}_n B_n C_{n-1}$ $A_n B_n \bar{C}_{n-1}$ $A_n B_n C_{n-1}$ $A_n \bar{B}_n C_{n-1}$
$\bar{C}_n$
C_n

Figure 18 Logic Charts and Connections for a Binary Adder

BINARY ADDER

In binary arithmetic, the addition rules are:

$$\begin{array}{rrrrrl} A = & 0 & 0 & 1 & 1 & \\ \underline{+B} = & \underline{+0} & \underline{+1} & \underline{+0} & \underline{+1} & \\ \text{Sum } \overline{S} = & 0 & 1 & 1 & 0 & \text{carry 1 to +1} \\ & & & & & \text{next higher column.} \end{array}$$

Each column of a binary number corresponds to a power of 2, instead of the familiar power of 10 used in decimal numbers. A sample addition of two numbers shows the binary number system and also a performance of the above rules.

sample:

$$\begin{array}{rrrr} A = & 0111 = & 0+4+2+1 = & 7 \\ \underline{+B} = & \underline{+1100} = & 8+4+0+0 = & \underline{+12} \\ \overline{S} = & \overline{10011} = & 16+0+0+2+1 = & 19 \end{array}$$

Therefore, to secure the proper sum digit (S) in a column, three quantities must be considered: the binary digit A, the binary digit B, and the carry digit C from the adjacent lower-ordered column. The complete addition rules, encompassing these three variables, can be briefly stated as follows. Create a sum digit equal to 1 for this column whenever the total number of 1's present in the three variables is an odd number. Create a carry digit equal to 1 for the next higher-ordered column whenever the total number of 1's present in the three variables equals or is greater than two.

These addition rules are shown displayed on the logic charts in Fig. 18 where the shaded combinations are those combinations that result in a sum digit or a carry digit equal to 1. Therefore, it requires two packages (a sum package and carry package) to add a column consisting of two binary digits. The T-PAC connections for a binary adder are shown in Fig. 18.

When the binary numbers are presented to the adder in serial form, only two T-PACs are required to accomplish the addition. This is due to the fact that each column is added in sequence, and the carry pulse is delayed 1 pulse period so that it plays a role in the addition of the next column.

When the binary numbers are presented to the adder in parallel form, two T-PACs per column are required to accomplish the addition. In this type of adder, the carry pulse from each column is presented to the next higher-order column so that the proper sum is created.

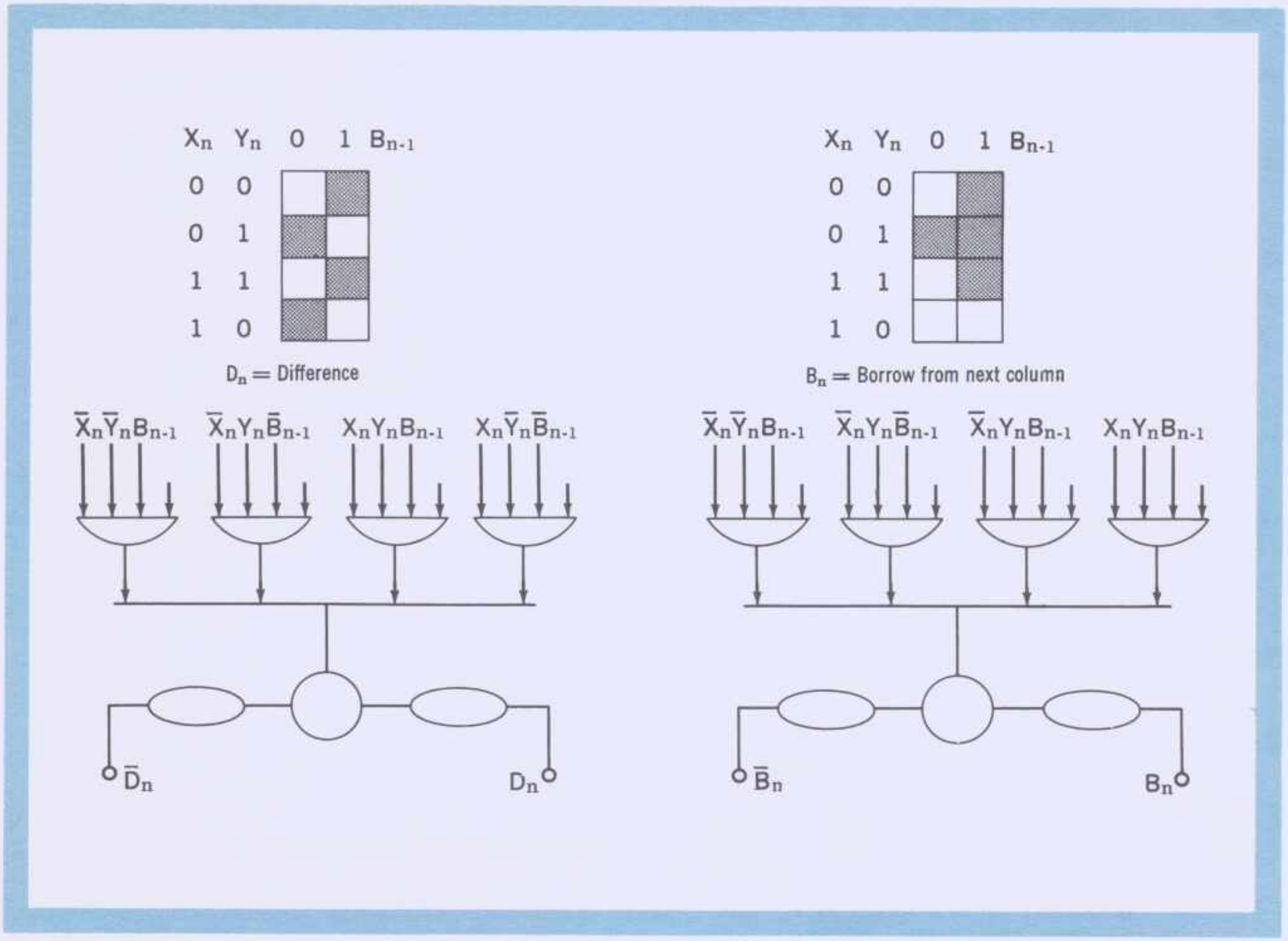

X_n Y_n 0 1 B_{n-1}
0 0
0 1
1 1
1 0
D_n = Difference
$\bar{X}_n\bar{Y}_nB_{n-1}$
$\bar{X}_nY_n\bar{B}_{n-1}$
$X_nY_nB_{n-1}$
$X_n\bar{Y}_n\bar{B}_{n-1}$
$\bar{D}_n$
D_n
X_n Y_n 0 1 B_{n-1}
0 0
0 1
1 1
1 0
B_n = Borrow from next column
$\bar{X}_n\bar{Y}_nB_{n-1}$
$\bar{X}_nY_n\bar{B}_{n-1}$
$\bar{X}_nY_nB_{n-1}$
$X_nY_nB_{n-1}$
$\bar{B}_n$
B_n

Figure 19 Logic Charts and Connections for a Binary Subtracter

BINARY SUBTRACTER

In binary arithmetic, the subtraction rules are:

X = 0 1 1 0
−Y = −0 −0 −1 −1
Difference D = 0 1 0 1 and borrow
1 from −1 next higher column

sample:

X = 1100 = 12
−Y = 0111 = −7
D = 0101 = 5

To secure the proper difference digit (D) in a column, three quantities, X, Y, and borrow digit B from the adjacent lower-ordered column must be considered. The complete subtraction rules, encompassing these three variables, can be briefly stated as follows: create a difference digit equal to 1 for this column whenever the total number of 1's present in the three variables is an odd number; create a difference digit whenever all three variables each equal 1 or whenever X is equal to 0 and Y or B equals 1.

These subtraction rules are shown displayed on the logic charts in Fig. 19. The T-PAC connections for a binary subtracter are also shown in Fig. 19. Serial subtraction requires two T-PACs, while parallel subtraction requires two T-PACs per column.

SERIAL UNITY ADDER

A serial unity adder, shown in Fig. 20, is a serial adder for the special case in which the augend is a binary number of any length, but the addend is always equal to unity. The advantage of this device is that it requires only one T-PAC. In Fig. 20, U is the unit add pulse, S is the sum pulse, C is the carry pulse, and R is the accumulated result.

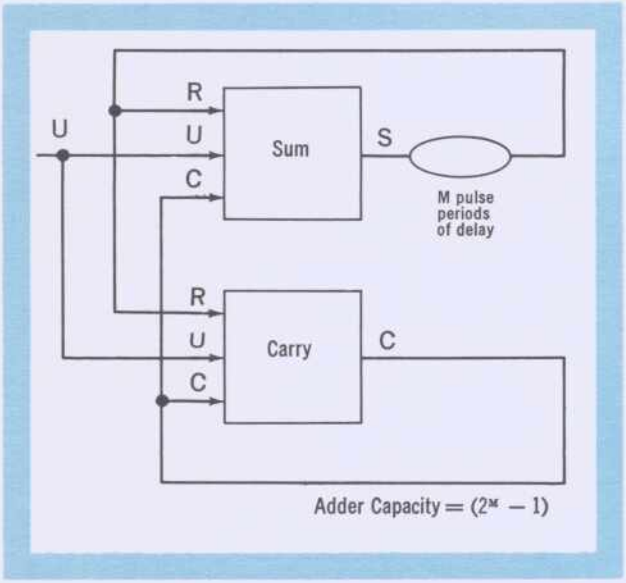


Figure 20 Block Diagram of Conventional Adder Connected as a Serial Unity Adder

Due to the limitation placed on the addend, it is impossible for a unit add pulse and a carry pulse to arrive simultaneously at the inputs to the sum and carry packages. The logic charts for the two functions are shown in Fig. 21. The darker areas depict the impossible combinations.

The chart for $\overline{S}$ is contrasted with the chart for C. The chart for $\overline{S}$ shows that if only those combinations containing R = 1 are used, a chart is obtained that is identical to the chart for C. Since the darker squares represent impossible combinations, these squares can be shaded or unshaded as desired. Therefore, C can be related to S as follows:

$$C = \overline{S} \cdot R$$

However, since the carry that plays a role in the addition of a certain column is always the result of a carry formed a pulse period earlier, it would be wise to acknowledge this timing by using C′, read "C delayed one pulse period", on the sum chart.

The last expression then reads $C' = \overline{S}' \cdot R'$. The sum chart employing C′ as a coordinate will now be replaced with a sum chart employing S′ and R′ instead. The result obtained shows that:

$$S = (U \cdot \overline{R}) \vee (\overline{S}' \cdot R' \cdot \overline{R}) \vee (\overline{U} \cdot R \cdot \overline{R}') \vee (\overline{U} \cdot R \cdot S')$$

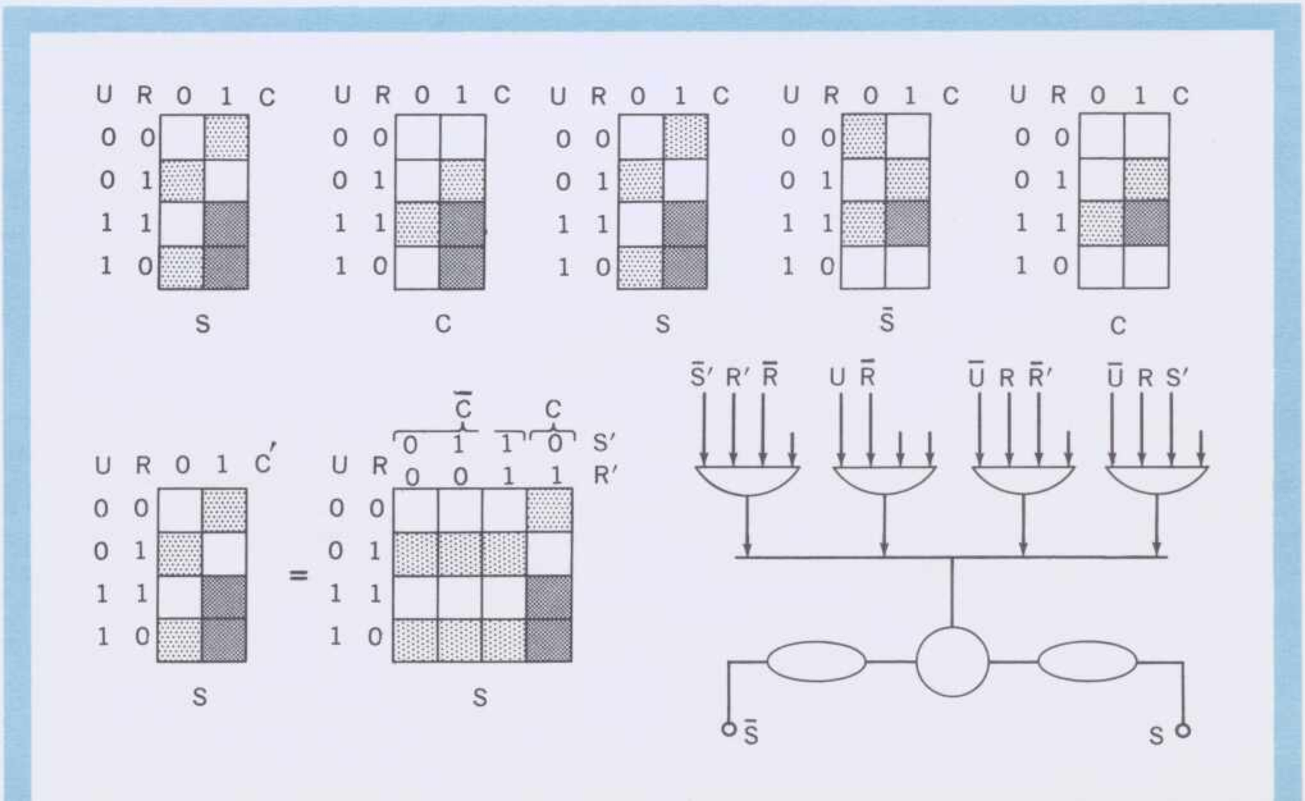
The connections for this single T-PAC are shown in Fig. 21.

SERIAL UNITY SUBTRACTER

Employing techniques similar to those of the serial unity adder, the following statement defines the difference pulse S:

$$S = (S' \cdot \overline{U} \cdot R) \vee (S \cdot \overline{R} \cdot \overline{R}') \vee (R \cdot R' \cdot \overline{U}) \vee (\overline{R} \cdot U)$$

Figure 21 Three Variable Logic Charts and Connections for single T-PAC Unity Adder



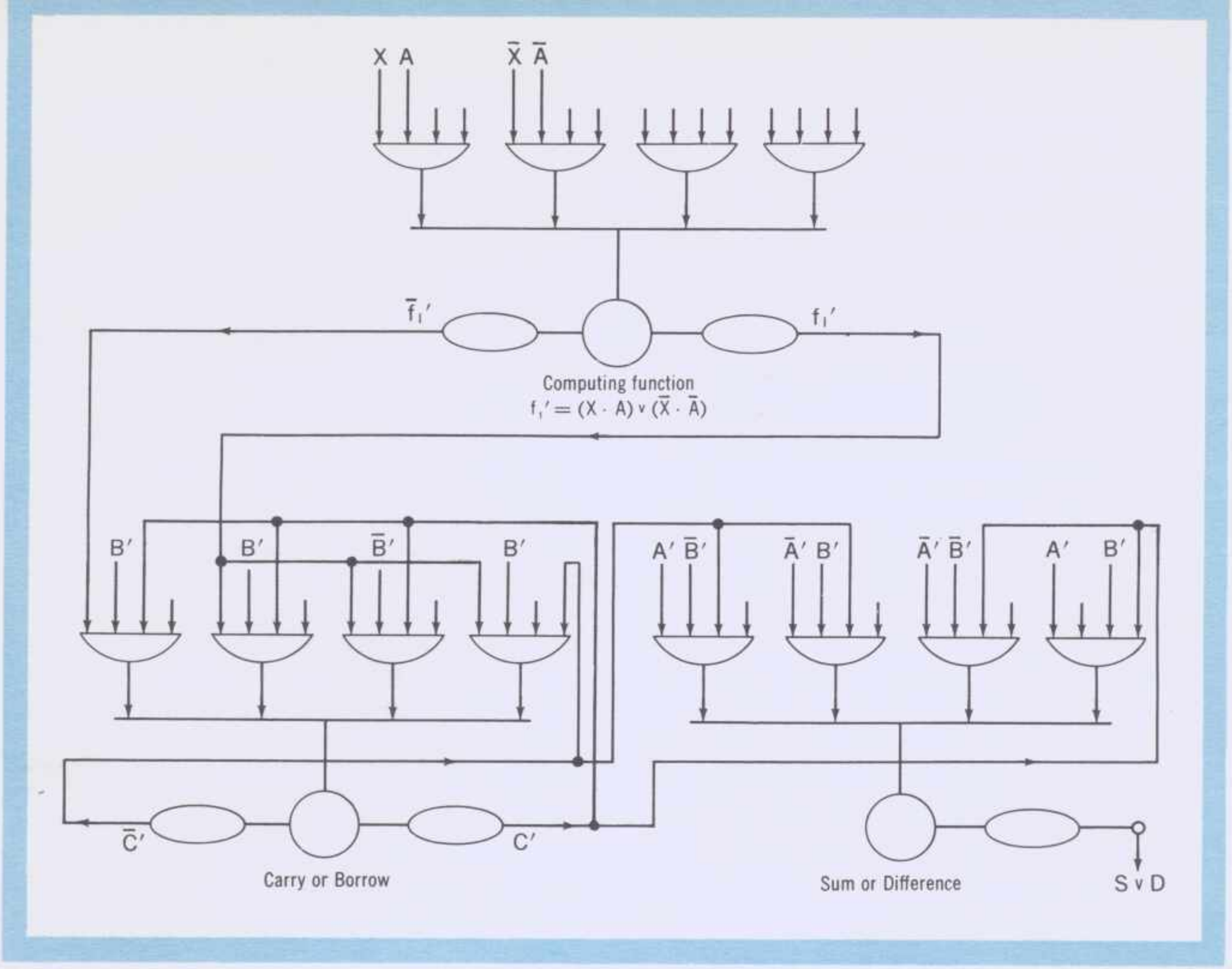


Figure 22 Combined Serial Adder-Subtracter with T-PACs

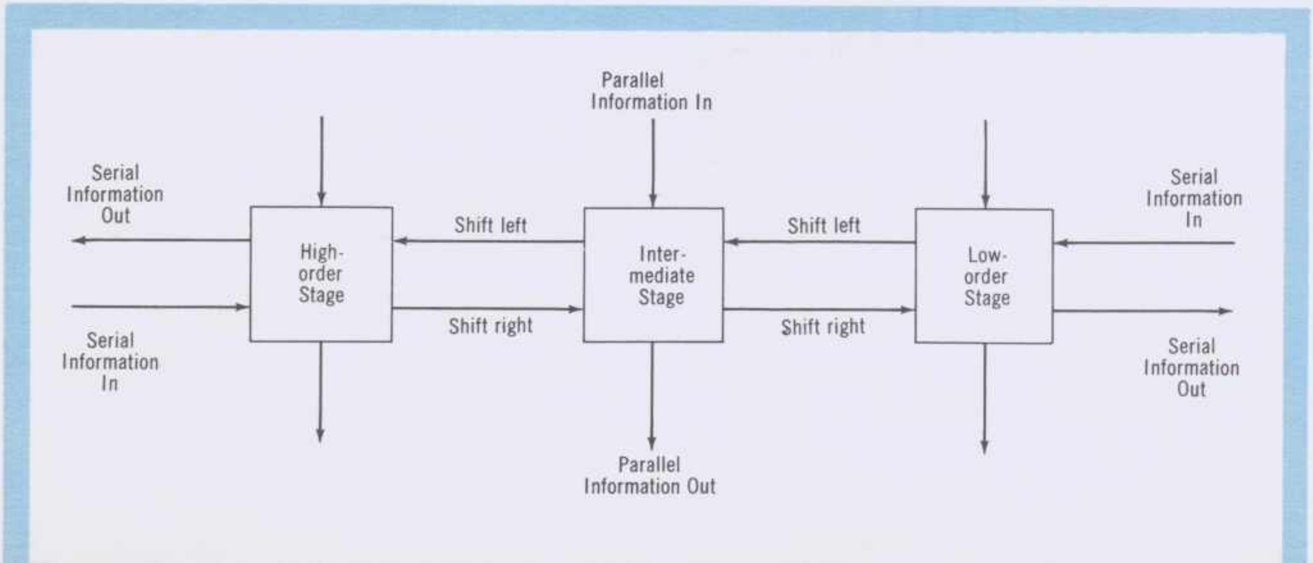
SERIAL ADDER-SUBTRACTER WITH T-PACs

For the utmost flexibility in binary computation, both addition and subtraction can be performed with three T-PACs, connected as shown in Fig. 22. An additional add-subtract command function $\underline{X}$ is required. This is in the form of a pulse train which causes addition to be performed when a pulse is present, and subtraction when a pulse is absent. The variables required for the serial adder-subtracter are:

A = Augend or minuend.
B = Addend or subtrahend.
C = Carry or borrow (delayed from preceding pulse period).
$\underline{X}$ = Add command.
$\overline{\underline{X}}$ = Subtract command.
D = Difference.
S = Sum.
$f_1 = (X \cdot A) \vee (\bar{X} \cdot \bar{A})$ computing function.

Using these variables, the serial adder-subtracter of Fig. 22 can be achieved with T-PACs.

Figure 23 Block Diagram of One Megacycle Shift Register



ONE MEGACYCLE SHIFT REGISTER

The 1-megacycle shift register is capable of doing the following functions up to the 1-megacycle rate:

1. Shift serial binary information either left or right.
2. Accept new information in serial form at either end of the shift register.
3. Accept new information in parallel form.
4. Transmit its contents in serial form from either end of the shift register.
5. Transmit its contents in parallel form from the shift register.

The block diagram of this register, employing only one T-PAC per stage, is given in Fig. 23.

let

W_p = Write in "parallel information".
P_n = nth bit of "parallel information".
I = Incoming bit of "serial information".
R_n = nth bit of register contents.
S_r = "Shift right" command.
S_l = "Shift left" command.

The T-PAC connections for a 1-Megacycle shift register are shown in Fig. 24.

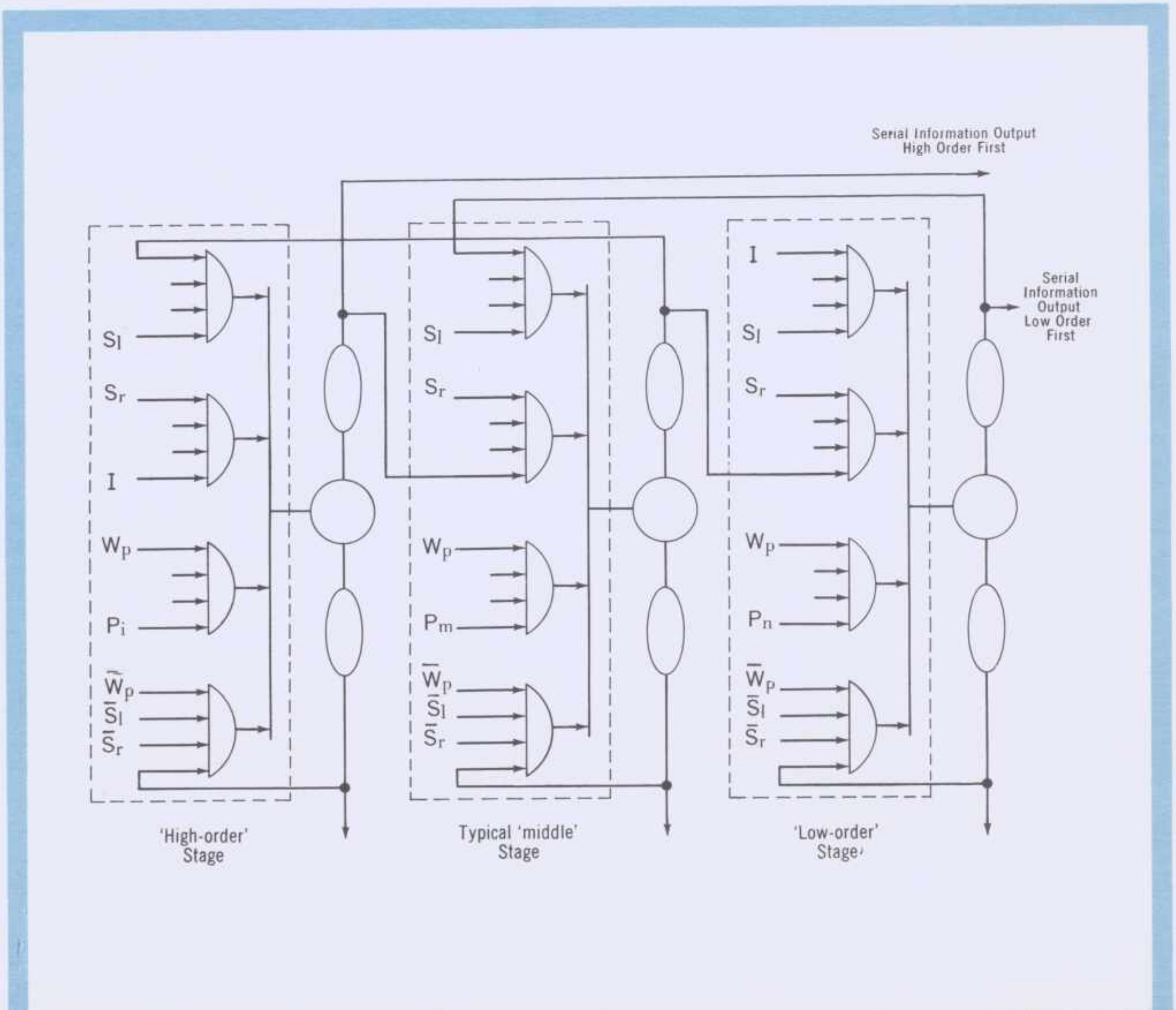
SERIAL COMPARATORS

Serial comparators, shown in Figs. 25, 26, and 27, are devices used to compare two serial binary numbers, A and B, and ascertain whether the numbers . . .

A and B are equal
A is greater than B
A is less than B

The solution for ascertaining if the numbers are equal requires special attention. If a disagreement is encountered during a comparison, subsequent agreement in later portions of the number or term must not conceal the fact that the quantities had

Figure 24 T-PAC Connections for One Megacycle Shift Register

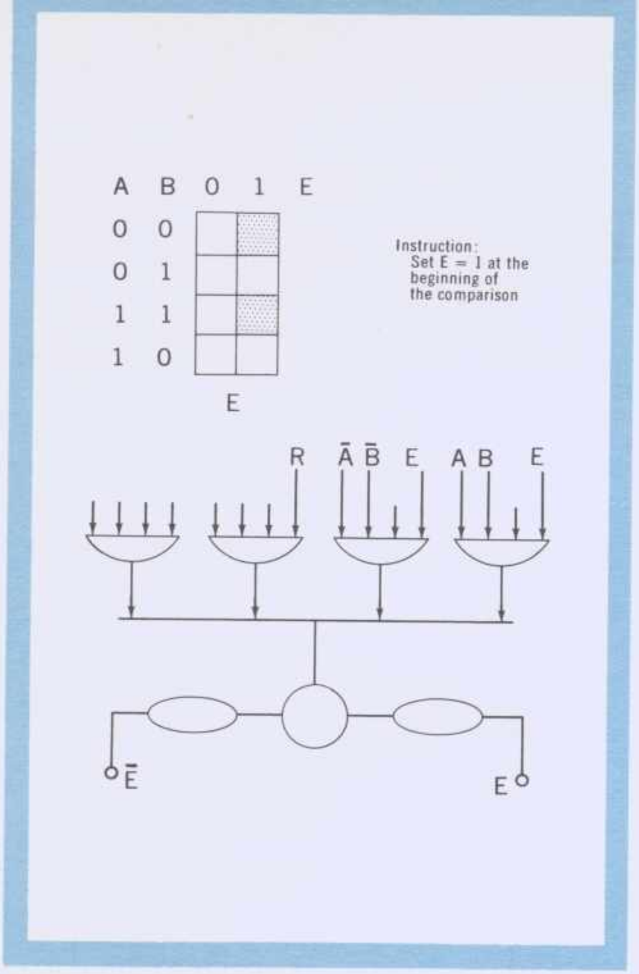


disagreed earlier. Therefore, the variable E is introduced and is defined such that E equals 1 so long as A equals B, but that E equals 0 after A and B are found not to be equal. The variable E is the output of the serial comparator, and is used also in stating the conditions necessary for the output.

The logic chart for this serial comparison function is shown in Fig. 25 with R being the initial reset pulse. If E has been set equal to 1 at the beginning of the comparison and the digits of A and B agree, E remains equal to 1. Should a disagreement between A and B appear, E becomes false (E = 0) and, although future sets of A and B digits agree, the false E prevents the successful comparison. On the other hand, if A and B agree throughout, then E will remain true all the time.

The definitions, logic charts, and T-PAC connections for "greater than" and "less than" serial comparators are shown in Figs. 26 and 27. Two particular conditions govern these comparisons:

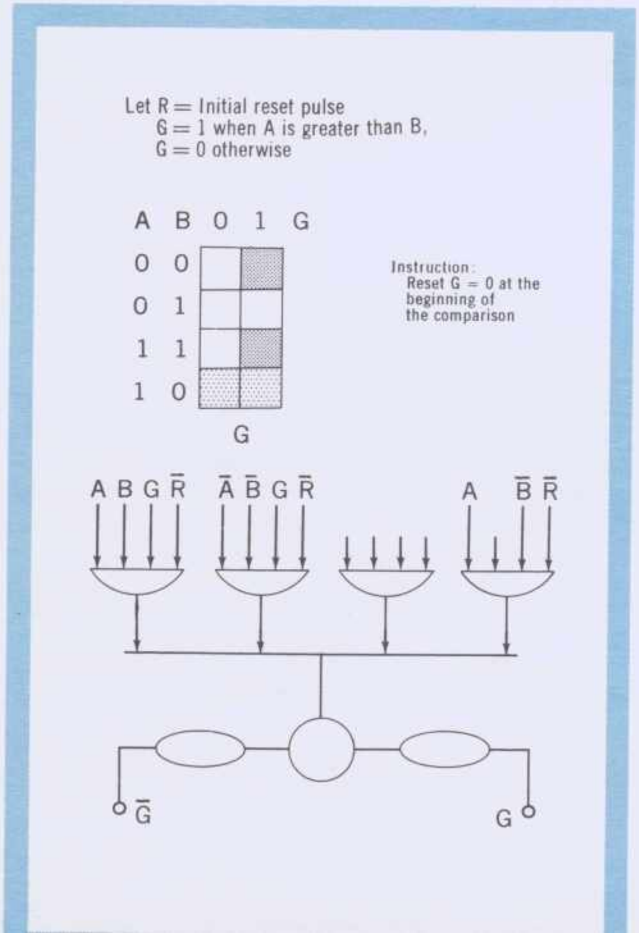
1. Agreements between A and B should not affect the results of these comparisons that are indicated by the darker areas of the charts.
2. Since low-order numbers are received first, the last disagreement in the comparison decrees the final result.



Instruction:
Set E = 1 at the beginning of the comparison

Figure 25 Logic Chart and Connections for Equality Comparator

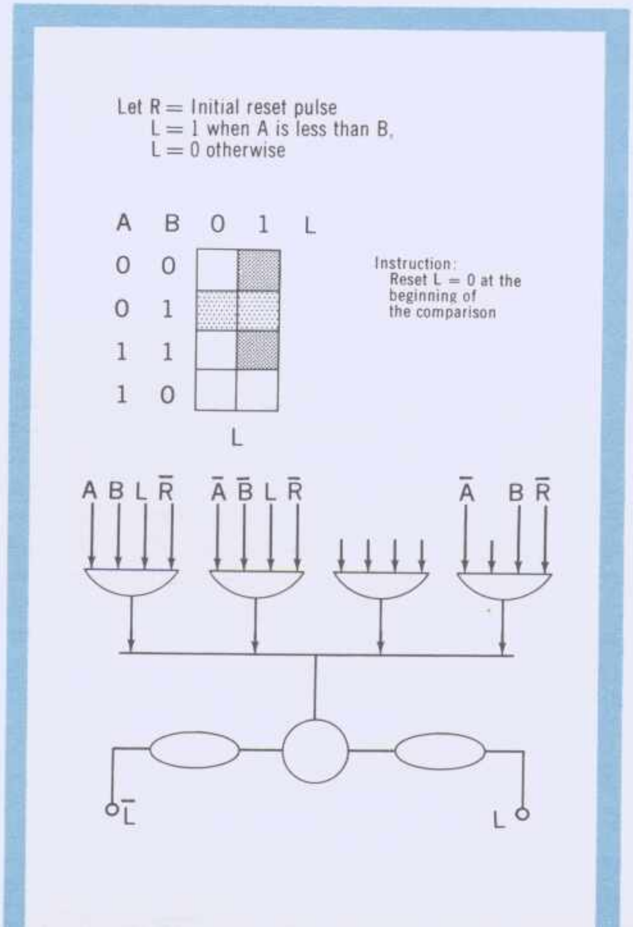
Figure 26 Logic Chart and Connections for 'greater than' Comparator



Let R = Initial reset pulse
G = 1 when A is greater than B,
G = 0 otherwise

Instruction:
Reset G = 0 at the beginning of the comparison

Figure 27 Logic Chart and Connections for 'less than' Comparator



Let R = Initial reset pulse
L = 1 when A is less than B,
L = 0 otherwise

Instruction:
Reset L = 0 at the beginning of the comparison

SERIAL GRAY CODE TO BINARY CODE CONVERTER

The Gray code, used in shaft-to-digital converters, is a reflected code: i.e., a code in which only one column changes value every time the number is increased one unit. Since the Gray code does not lend itself to computation, a conversion to binary code is frequently employed. The binary code and its equivalent Gray code are shown in the table below:

Decimal	Binary				Gray			
	B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0

Let

B_n = nth Binary digit

G_n = nth Gray code digit

The serial conversion is accomplished by treating the serial Gray-coded number high-order first and implementing the statement:

$$B_{n-1} = (\bar{B}_n \cdot G_{n-1}) \vee (B_n \cdot \bar{G}_{n-1})$$

T-PAC connections for this converter are shown in Fig. 28.

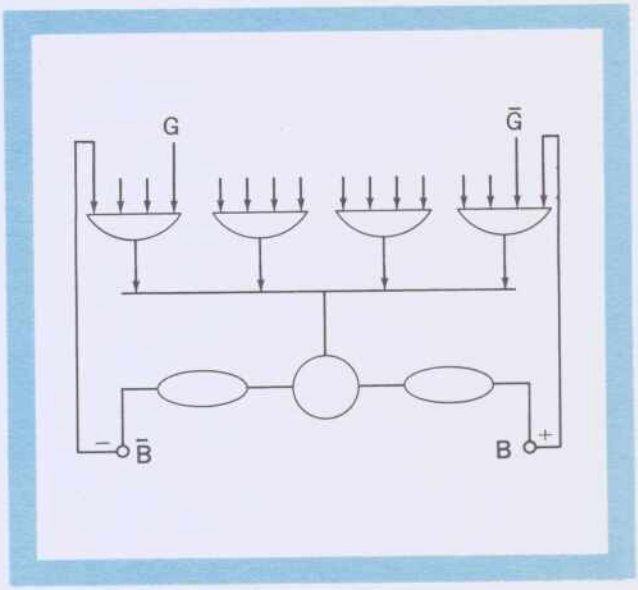
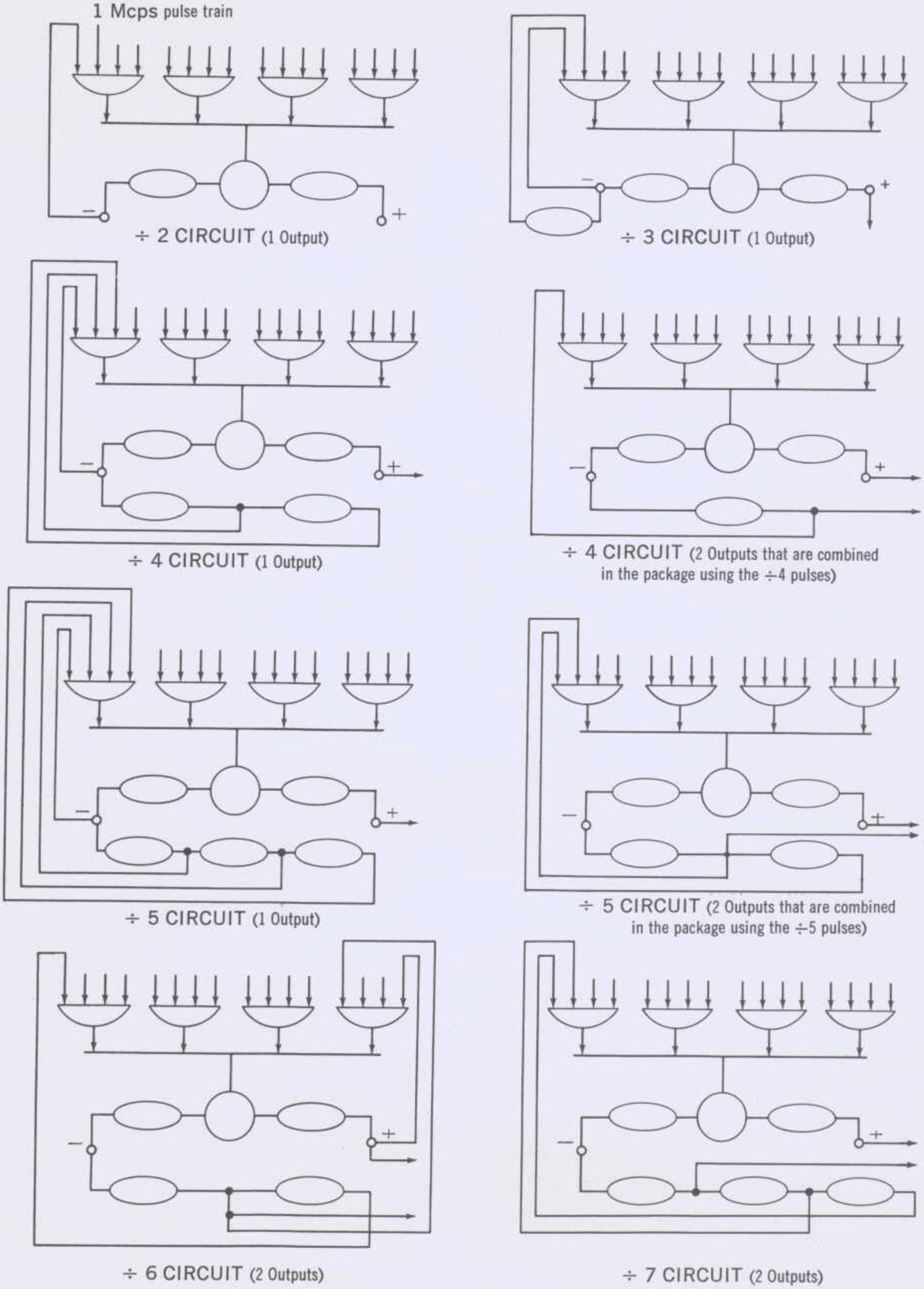


Figure 28 T-PAC Connections for Serial Gray Code to Binary Code Converter

PULSE PATTERN GENERATORS
(self-starting, free-running)

Pulse pattern generators are devices which generate sequential configurations of pulses with a particular periodicity or repetitiveness. Such devices are found to be extremely useful to the digital designer for a variety of applications. These include clock frequency division, the generation of complex timing control signals for arithmetic processes, the generation of pseudo-random numbers, etc. T-PACs are ideal logical elements for the implementation of pulse pattern generators. With a single T-PAC Model LE-10 and a maximum of ten unit delays, repetitive pulse patterns up to 63 microseconds in length (one microsecond equals one pulse time) have been designed. With the use of relatively few additional T-PACs periodicities many thousand bits in length can be achieved.

Fig. 29 illustrates the usage of T-PACs for very short period pattern generators or frequency dividers. Patterns of greater length are created by the same technique of logical feedback.



÷ 2 CIRCUIT (1 Output)

÷ 3 CIRCUIT (1 Output)

÷ 4 CIRCUIT (1 Output)

÷ 4 CIRCUIT (2 Outputs that are combined in the package using the ÷4 pulses)

÷ 5 CIRCUIT (1 Output)

÷ 5 CIRCUIT (2 Outputs that are combined in the package using the ÷5 pulses)

÷ 6 CIRCUIT (2 Outputs)

÷ 7 CIRCUIT (2 Outputs)

Figure 29 Seif-starting Unambiguous Pulse Pattern Generators

ONE-SHOT PULSE PATTERN GENERATORS

A single pulse input can be delayed and regenerated to produce a unique self-quenching pattern. By sensing particular code sequences in this pattern it is possible to obtain single or multiple pulse outputs delayed as desired. Fig. 30 illustrates such a pattern generator for obtaining delays up to 29 pulse periods. The output package senses on $\bar{A}_1 \cdot \bar{A}_2 \cdot \bar{A}_3 \cdot \bar{A}_4 \cdot A_5$. The reason for this can be seen from inspection of the one-shot pattern generated by this wiring logic which is as follows:

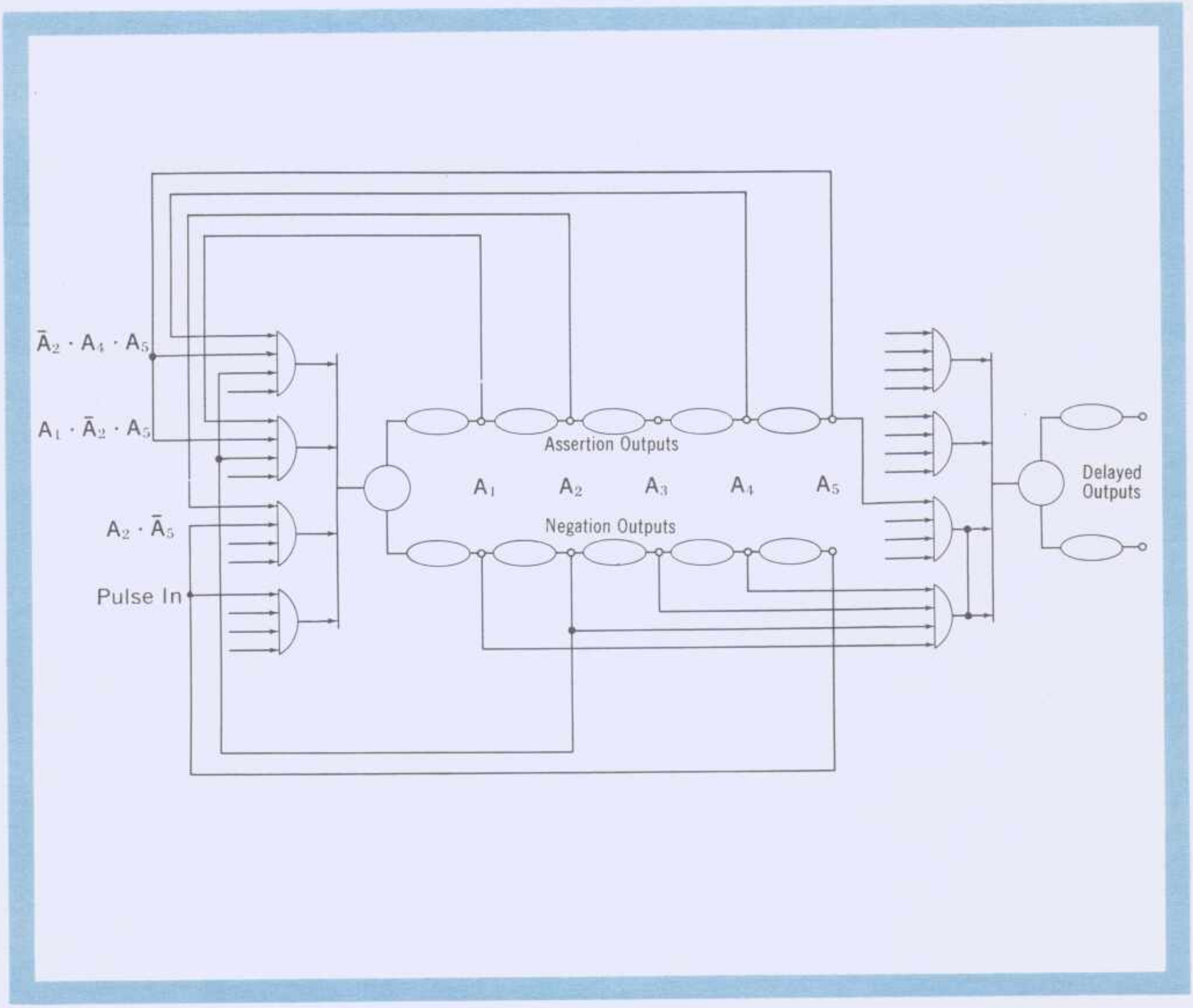
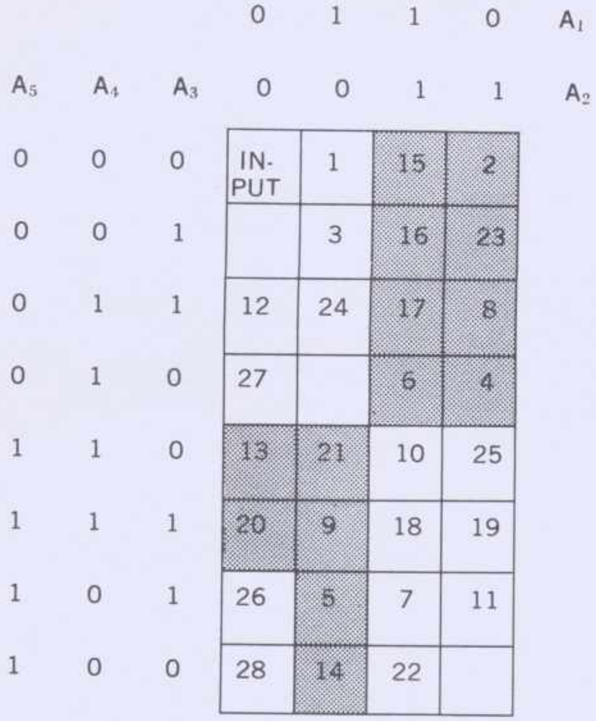


Figure 30 Wiring Diagram for 29 Pulse Period Delay

timing ─▶	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	28	29
pattern ─▶	0	1	0	1	0	1	1	1	0	1	1	0	0	0	1	1	1	1	1	0	0	1	1	0	1	0	0	0	0	0
input ─▶	1																											SELF-QUENCHING		
output ─▶																														1



			0	1	1	0	A_1
A_5	A_4	A_3	0	0	1	1	A_2
0	0	0	IN-PUT	1	15	2	
0	0	1		3	16	23	
0	1	1	12	24	17	8	
0	1	0	27		6	4	
1	1	0	13	21	10	25	
1	1	1	20	9	18	19	
1	0	1	26	5	7	11	
1	0	0	28	14	22		

Figure 31

This pattern was achieved by the implementation of the truth table of Fig. 31 for the five binary variables A_1, A_2, A_3, A_4, and A_5 which at all times represent a five pulse period sequential view of the output of the initial T-PAC of Fig. 30. The numbers in the squares represent the sequence timewise of the occurrence of the five variables.

EDITOR'S NOTE: *A future publication by 3C will attempt to treat in more detail the logical design techniques whereby free-running and self-quenching pattern generators of specific configuration and periodicity are achieved.*

HISTORY

COMPUTER CONTROL COMPANY Inc. was organized in 1952. From its inception two of the major goals of the company have been:

1. Research, design, development, and marketing of an advanced line of high quality, ultra-reliable digital building blocks.
2. Quality engineering of special purpose digital systems.

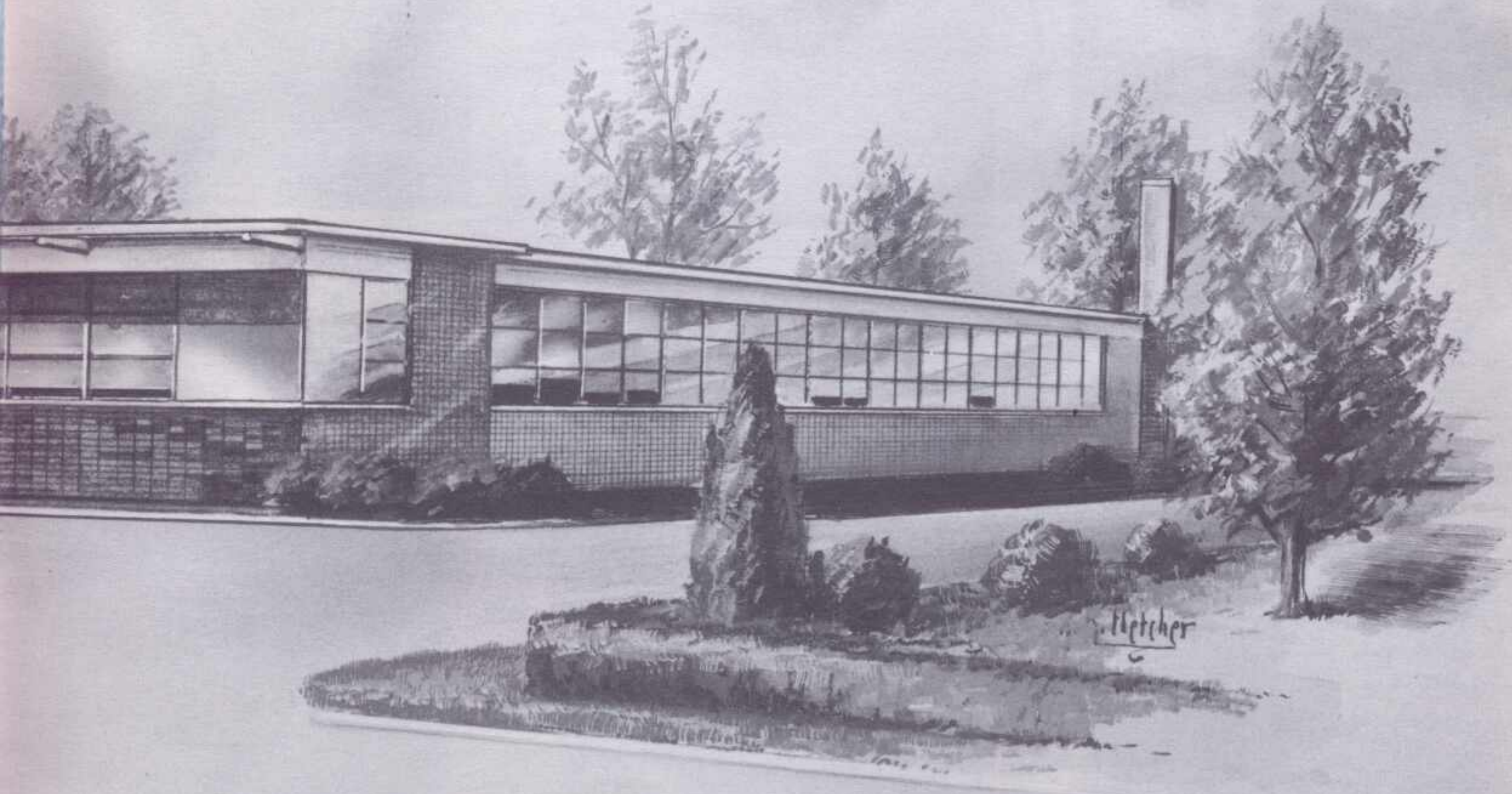
In 1953 an additional major objective was achieved with the formation of a Mathematical Services Division. This activity is staffed by a highly competent group of mathematicians who offer the following services to government and industry:

3. Problem analysis, programming, coding and machine computation.
4. General mathematical analysis and consultation.

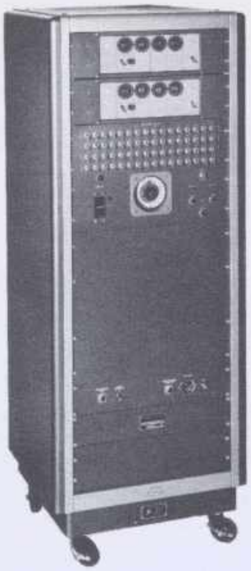
To best serve its expanding list of customers the company has been organized about two geographical divisions. The Eastern Division is now located in a modern well-equipped manufacturing plant and laboratory in Framingham, Massachusetts, twenty miles west of Boston. The Western Division utilizes similarly well-equipped up-to-date facilities at its strategic location in West Los Angeles.

EASTERN DIVISION

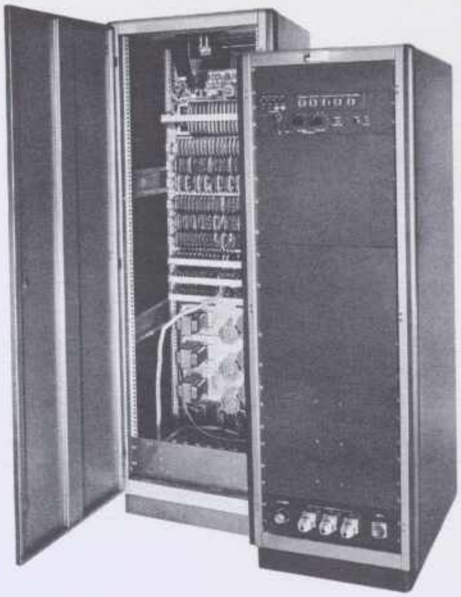
WESTERN DIVISION



Digital Director for Automatic Milling Machine

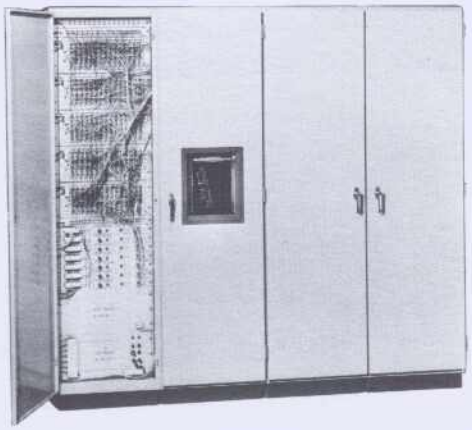


Information Transmission Comparator

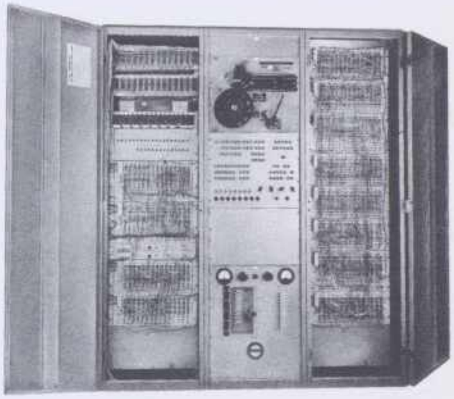


Radio Telescope Director
(FRONT AND REAR VIEW)

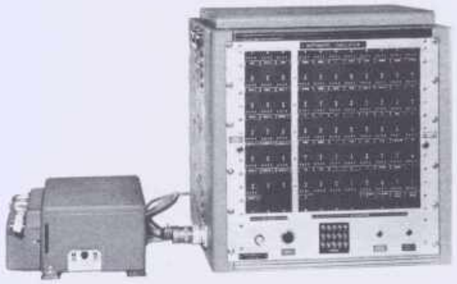
Automatic Test Equipment



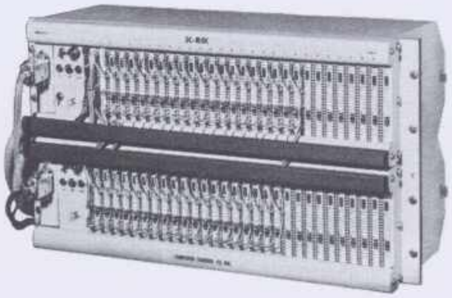
Information Retrieval System



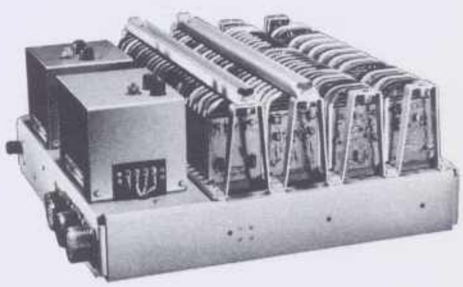
Computer Input-Output System



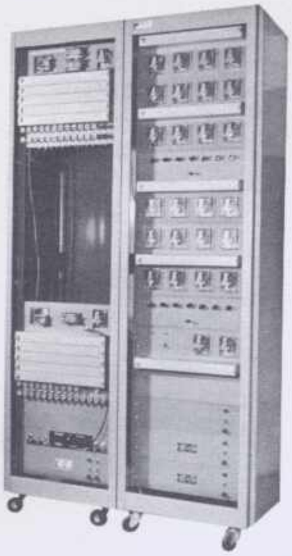
Automatic Decoder and Item Tabulator



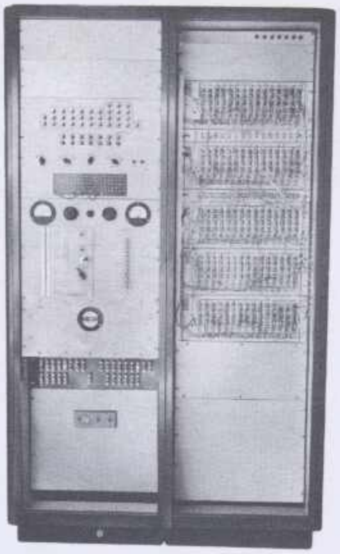
Special Purpose Shift Register



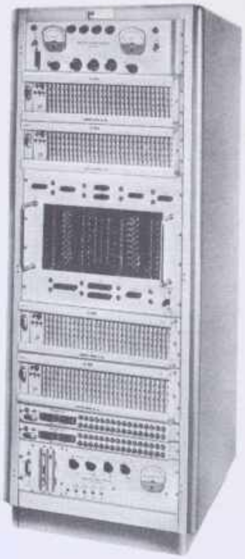
Digital Comparator



Radio Telescope Readout System



Range Time Coder



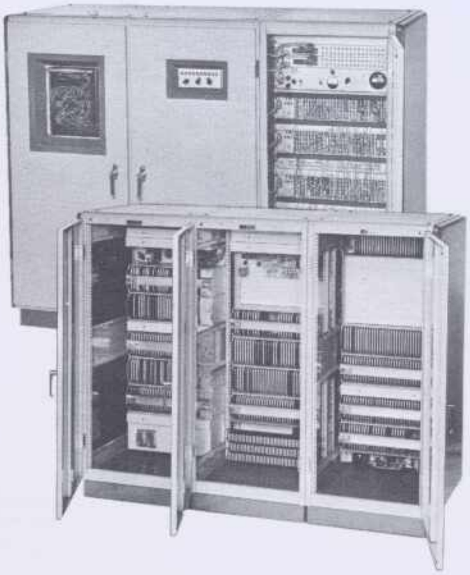
High Speed Random Access All-Transistor Magnetic Core Memory

SYSTEMS

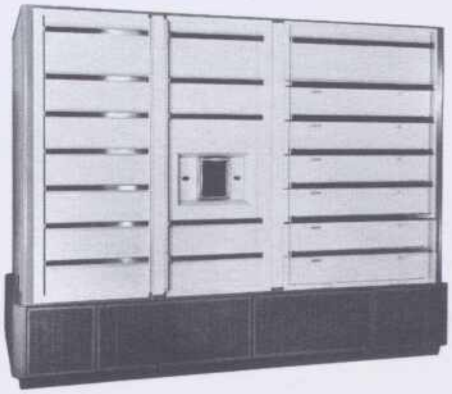
These photographs show some of the custom engineered special purpose digital equipments and computers manufactured by 3C for a variety of customers and for a variety of applications. We regret that many other systems of unique interest cannot be shown because of security classifications.

Information Retrieval System

Digital Monitoring System



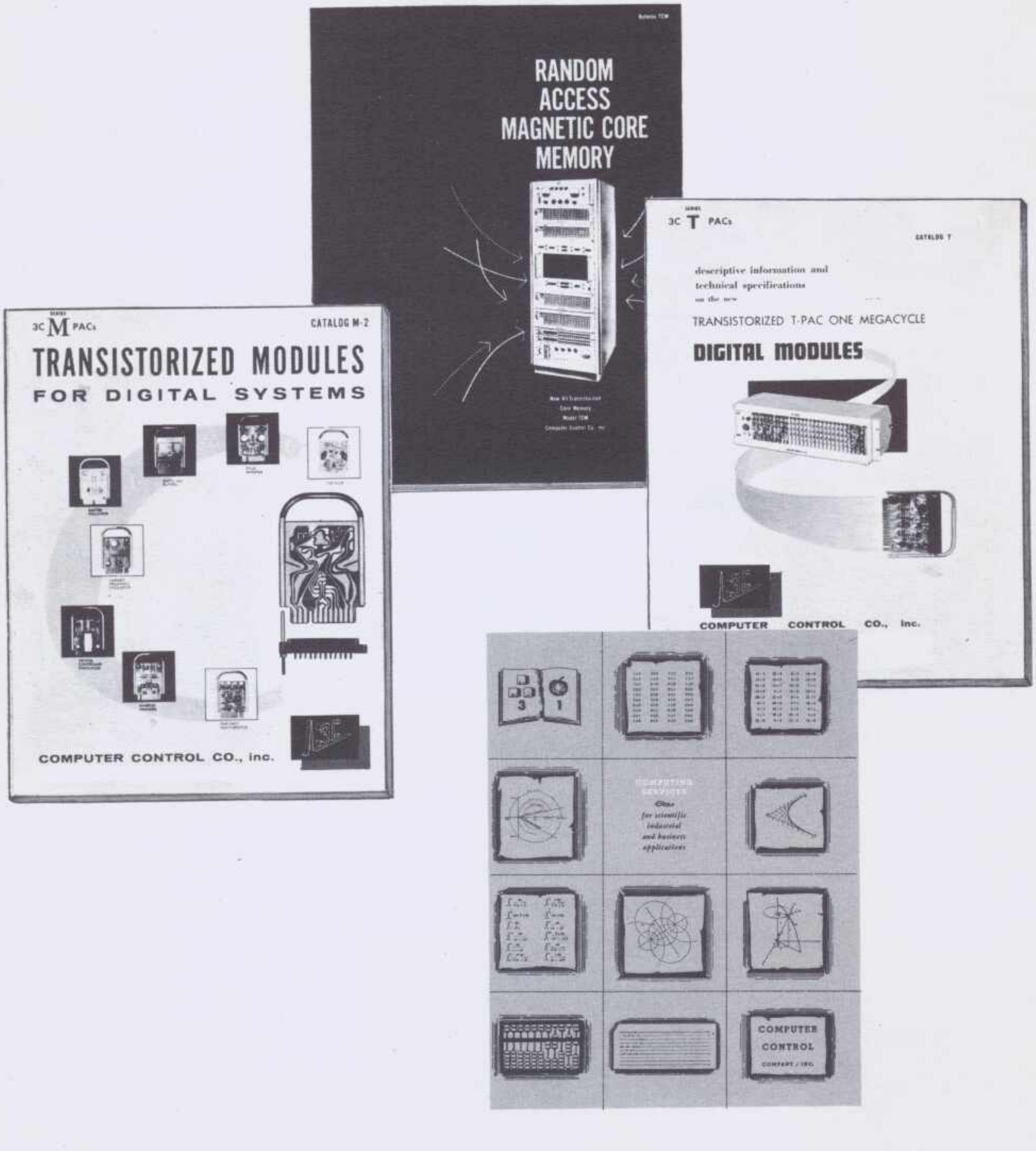
Information Retrieval System
(FRONT AND REAR VIEW)



Information Retrieval System

PRODUCTS

3C's transistorized digital plug-in packages and other products and services are described in detail in the following publications. Product specifications and prices are included.



RANDOM ACCESS MAGNETIC CORE MEMORY

3C T PACs

CATALOG T

descriptive information and technical specifications on the new

TRANSISTORIZED T-PAC ONE MEGACYCLE

DIGITAL MODULES

COMPUTER CONTROL CO., Inc.

3C M PACs

CATALOG M-2

TRANSISTORIZED MODULES FOR DIGITAL SYSTEMS

COMPUTER CONTROL CO., Inc.

COMPUTING SERVICES

for scientific industrial and business applications

COMPUTER CONTROL COMPANY / INC.

20 page Catalog M and supplementary bulletins describe the 3C-PAC series M-family of compatible plug-in modules

16 page Catalog T and supplementary bulletins describe the 3C-PAC series T-family of compatible plug-in modules

8 page Bulletin TCM describes 3C's all transistorized high-speed random access ferrite core memories

4 page Computing Services brochure lists the capabilities and available services of the Mathematical Services Division

Available soon: Brochure describing SPEC, 3C's stored program educational computer. SPEC is designed to be used as an educational tool and a general purpose computer. It may also be expanded for use as a digital differential analyzer.

AVAILABILITY OF 3C SERVICES

To solve your digital problems COMPUTER CONTROL COMPANY offers the services of its staff of . . .

- Circuit designers
- Logical designers
- Systems engineers
- Mathematical analysts, computer programmers and coders

Our complete flexibility permits a variety of workable arrangements. We can share your problems to whatever extent you wish, ranging from minor consultation to complete design, development, and construction.

Write, wire, or phone us for further information.

COMPUTER CONTROL COMPANY, INC.

EASTERN DIVISION: 983 CONCORD STREET • FRAMINGHAM, MASSACHUSETTS
BOSTON TELEPHONE: CEdar 5-6220
WESTERN DIVISION: 2251 BARRY AVE. • WEST LOS ANGELES 64, CALIFORNIA
BRadshaw 2-9135/GRanite 8-0481

ADDENDUM AND ERRATA
FOR
Symbolic Logic, Boolean Algebra and the Design of Digital Systems

By the Technical Staff of COMPUTER CONTROL COMPANY INC.

(First edition 20M-759-BA)

A surprisingly large number of readers have written to 3C expressing their appreciation of Symbolic Logic. We are very happy to hear from them. Some have thoughtfully offered constructive suggestions where corrections and additions to the text would make it even more helpful. These suggestions have been reviewed and are printed below.

Page 4
Left column, ALGEBRA, paragraph 3 add . . .

$$ab = ba$$

4. Distributive Law

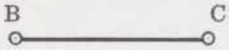
$$a\,(b + c) = ab + ac$$

Page 4
Right column, SYMBOLIC LOGIC, paragraph 3 add . . .

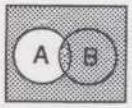
4. Distributive Laws

$$a\,(b \vee c) = ab \vee ac$$
$$a \vee (bc) = (a \vee b)\,(a \vee c)$$

Page 6
ELECTRONIC IMPLEMENTATION column 4 line 4 read . . .



Page 6
VENN DIAGRAM, column 2 line 8 read . . .



Page 8
Right Column, ALTERNATE AND PREFERABLE STEP 2, omit last two lines beginning "Application of step 3" etc.

Page 9
Figure 7, second chart from left, change caption to read . . . $\bar{A}$

Page 9
Figure 7, fifth chart from the left, change caption to read . . .

$$f_s = (A \cdot B) \vee [\bar{A} \cdot (B \vee C)] = B \vee (\bar{A} \cdot C)$$

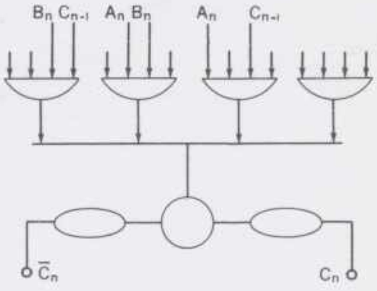
Page 9
Figure 7, caption (below blue outline) add . . .
Application of step 3 to the Chart will show $f_s = B \vee (\bar{A} \cdot C)$

Page 10
Figure 8, lower right hand gate, correct symbols to read . . .

$\bar{A}$
$\bar{B}$
$\bar{C}$
$\bar{D}$

Page 16
Figure 16, lower left diagram, read . . . Dc $\bar{D}c$

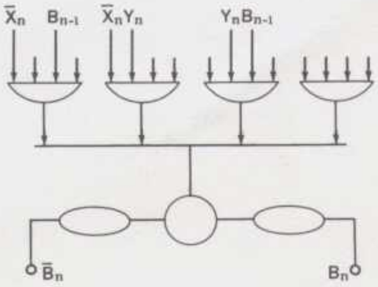
Page 18
Figure 18, lower right hand diagram, minimize to read . . .



Page 18
Left column, BINARY ADDER, lines 5 and 6, change to read...

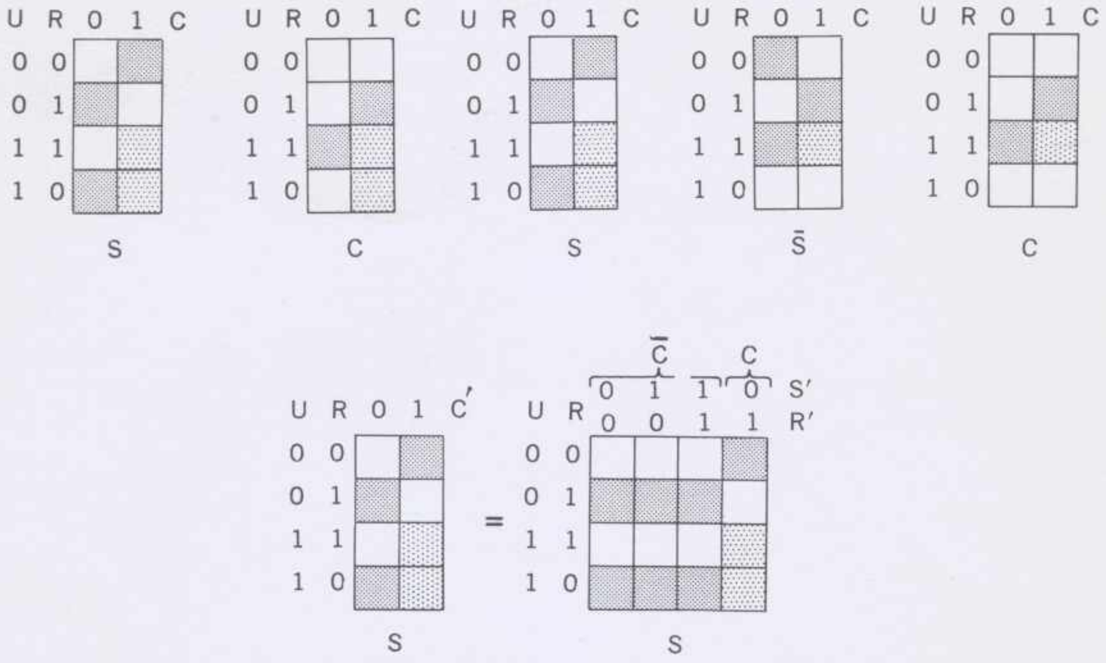
Sum S = 0 1 1 0 carry 1 to next higher column.

Page 19
Figure 19, lower right-hand diagram, minimize to read . . .

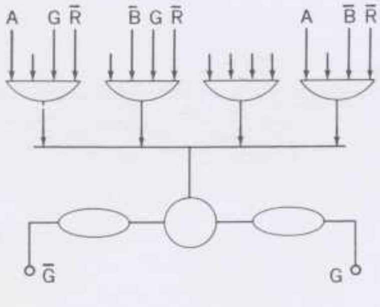


Page 19
Left column, BINARY SUBTRACTER, lines 4 and 5, change to read . . .
1 from next higher column

Page 20
Figure 21, simplified diagrams read . . .

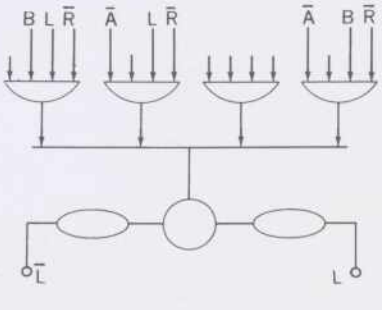


Page 23
Figure 26, lower diagram, minimize to read . . .



Page 23
Right column, paragraph 3, sub paragraph 1, line two, read . . .
"affect the results of past comparisons that" etc.

Page 23
Figure 27, lower diagram, minimize to read . . .



COMPUTER CONTROL COMPANY, INC.
983 CONCORD STREET • FRAMINGHAM • MASSACHUSETTS
2251 BARRY AVENUE • LOS ANGELES 64 • CALIFORNIA