AN ABSTRACT OF THE THESIS OF

SUNG CHIAO HU for the M.S. in ELECTRICAL AND ELECTRONICS ENGINEERING
(Name) (Degree) (Major)

Date thesis is presented May 12, 1967

Title TERNARY DIGITAL SYSTEMS

Abstract approved Redacted for Privacy
Professor Lewis N. Stone

In view of recent developments in large-scale, complex digital systems, it is of interest to broaden the study of two-leveled logical systems to the study of three-leveled systems. This paper introduces the ternary logic system and develops a design approach for ternary digital systems that is based on familiar binary techniques.

Ternary number systems are introduced and two base-conversion algorithms are given: one for integral numbers and the other for fractional numbers. Ternary arithmetic is similar to binary arithmetic or to decimal arithmetic. Four algebraic operations, Cycling, Negation, And, and Or, are then defined and the algebra is systematically developed through postulates and theorems. The algebraic method of minimization is difficult for a large number of variables or terms, and the map method is impracticable for more than three variables. Hence a programmable minimization method is developed by analogy with the Quine-McCluskey method for binary minimization.

Diode-transistor schemes of circuit realization are presented. In these, the idea is to use p-n-p and n-p-n transistor pairs to provide the three different voltage levels desired. Tristable devices using three Cycling-gates are also described. A core storage element employs two differently oriented cores and provides one ternary digit of storage. Additional algebraic operations that are used in the current literature are given in the Appendix.

TERNARY DIGITAL SYSTEMS

by

SUNG CHIAO HU

A THESIS

submitted to

OREGON STATE UNIVERSITY

in partial fulfillment of
the requirements for the
degree of

MASTER OF SCIENCE

June 1967

APPROVED:

Redacted for Privacy

Professor of Electrical and Electronics Engineering

in charge of major

Redacted for Privacy

Head of Department of Electrical and Electronics Engineering

Redacted for Privacy

Dean of Graduate School

Date thesis is presented May 12, 1967

Typed by Erma McClanathan for [illegible]

ACKNOWLEDGMENTS

The writer is indebted to Professor Louis N. Stone for his valuable help and comments in the writing of this thesis. The writer also wishes to thank Professor Robert A. Short who read the manuscript and offered many constructive suggestions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TERNARY DIGITAL SYSTEMS

I. INTRODUCTION

The ternary digital system is one possible form of digital system. Although binary switching theory has been used in almost every digital system developed so far, this is mainly due to the fact that basic switching elements in common use are two-state devices and that design techniques for the binary system are well-developed. However, it is useful to broaden the study of two-leveled logical systems to that of three-leveled systems. The motivation for such an extension is based upon the following reasons:

1. Some of the operating systems in use actually involve three discrete states in logical relations. For example, industrial control systems can be most directly handled with mathematical models of three-valued logical systems.

2. Three-valued systems offer an alternative approach to ameliorate some of the complicated problems which arise in two-valued systems.

3. The advance of electronic technology should make reliable three-state devices available in the not-too-remote future. It is shown in (8) that if the amount of the equipment required is proportional to the

number base, then the most efficient base to use is base three.

4. Ternary systems can provide several advantages over binary systems such as speed, cost, space, weight, etc. Also, they have been proposed to enhance reliability by using the third value to indicate error situations.

5. In the fields of communication and data transmission, the greater base implies more accurate transmission per transmitted digit. In this regard, the study of the ternary system can be thought of as a start for systems utilizing even higher bases.

With a three-valued system, three-valued switching functions must be used. To this end, a number of authors have been concerned recently with the design of ternary switching circuits. A ternary number system was introduced by Morris (9). A comparison of base three vs. base two systems is given by Santos (14). Lee and Chen (5), Muehldorf (10), and Vacca (16) consider the algebraic aspects of the ternary system. Lowenschuss (6) and Muehldorf (9) have suggested the application of the Rutz transistor to the design of ternary circuits. Other ternary devices also have been suggested in the literature (1), (4), (13). Diode-transistor schemes were introduced by Hallworth and Heath (2) and Santos

(15). Other papers on switching theory and logical design include (6), (7), (10), and (17). An extension of the application of binary devices and Boolean algebra to the realization of three-valued logic circuits was recently introduced by Pugh (12).

This paper develops a design theory for a ternary digital system that is based on familiar binary algebra, including the simplification and circuit design techniques of binary systems. The main text is divided into four parts. The first part introduces the ternary number systems and arithmetic; the second part discusses the algebraic properties of the ternary system; the third part gives three minimization methods for ternary functions; the last part presents some techniques appropriate for hardware realizations.

II. TERNARY NUMBER SYSTEMS AND ARITHMETIC

The most commonly used number system in computers and other switching circuits is the well-known binary number system. This is due to the fact that most physical devices in common use exhibit two easily distinguished states. Ternary (base 3) number systems, however, have been shown to be the most efficient number system if the assumption that the required equipment is proportional to the number base is met. In this chapter, the ternary number system will be introduced. Algorithms for base conversions will be developed and illustrated by means of examples. Finally, ternary arithmetic will be defined.

A. Ternary Number Systems

Definition 2.1. A number N is said to be in ternary form if and only if

$$N = \sum_{i=-n}^{m-1} t_i 3^i$$

$$= t_{m-1}3^{m-1}+t_{m-2}3^{m-2}+\cdots+t_1 3^1+t_o 3^o+t_{-1}3^{-1}\cdots+ t_{-n}3^{-n}$$

where m is the number of integral digits,

n is the number of fractional digits,

and $t_i \in \{0,1,2\}$.

In general, we simply write

$$N = t_{m-1}t_{m-2}\cdots t_1t_o.t_{-1}\cdots t_{-n}$$

where the "." is called the "ternary point," and t_i is a "ternary digit."

Definition 2.2. The complement of a ternary digit, t, denoted by $\bar{t}$, is the difference between that digit and the number 2, thus

$$\bar{t} = 2 - t.$$

There are three ways to represent a negative ternary number, and these are analogous to the usual binary conventions.

Negative of N	$= -N$	sign magnitude
	$= 3^m-N$	3's complement
	$= 3^m-N-L$	2's complement.
		L represents one in the least significant place of N.

B. Base Conversions

Since most present digital devices utilize the binary number system, the conversion between a base 2 number and a base 3 number is very important. The conversion between ternary and decimal systems is, of course, also of great interest. Table I defines such conversions on a digit basis.

Table I. Base Conversions Between (a) Base 2 Digits and Base 3 Digits, (b) Base 3 Digits and Base 10 Digits.

(a)

Base 2	Base 3
0 0	0
0 1	1
1 0	2

(b)

Base 3	Base 10
0 0 0	0
0 0 1	1
0 0 2	2
0 1 0	3
0 1 1	4
0 1 2	5
0 2 0	6
0 2 1	7
0 2 2	8
1 0 0	9

Two algorithms are now presented for conversions between any two number systems. One is for integral numbers, and the other is for fractional numbers.

Algorithm A: (for base conversion of integral numbers)

1. Express the "new base," b_2, in terms of the "old base," b_1.
2. Divide the "old number" and its successive quotients by the number found in step one until

the quotient is zero. Name the successive remainders as r_1, r_2, r_3, $\cdots$. Note that the remainders are still in the "old base."

3. If $b_2 < b_1$, then the remainders obtained in step two are in correct form. If $b_2 > b_1$, then the remainders must be converted into the "new base."
4. The remainders in correct form are then taken in reverse order to produce the "new number."

Example: Convert the number 101001 in base 2 to its representation in base 3.

1. 3 is represented by 11 in base 2
2. $11\,\underline{\big|101001}$ -------- 10 (r_1)

   $11\,\underline{\big|1101}$ -------- 1 (r_2)

   $11\,\underline{\big|100}$ ------- 1 (r_3)

   $11\,\underline{\big|1}$ ------- 1 (r_4)

   0
3. Since $3 > 2$, the remainders are converted from base 2 to base 3

   r_1: $10 \longrightarrow 2$

   r_2: $1 \longrightarrow 1$

   r_3: $1 \longrightarrow 1$

   r_4: $1 \longrightarrow 1$
4. The number in base 3 form is

   $N_3 = r_4 r_3 r_2 r_1 = 1112.$

Algorithm B: (for base conversion of fractional numbers)

1. Express the "new base," b_2, in terms of the "old base," b_1.
2. Multiply the given fractional number by the number found in step one. The resulting integral part is designated as I_1 and the fractional part as F_1.
3. Again multiply the fractional part, F_1, by the number found in step one. The resulting integral part is designated as I_2 and the fractional part as F_2.
4. Continue the process in this fashion until F_i becomes zero or i reaches the desired degree of significance.
5. If $b_2 < b_1$, then no conversions are made to the successively obtained integral parts. If $b_2 > b_1$, each integral part is converted into the "new base."
6. The "new number" is formed by taking integral parts in right order, i.e., $I_1 I_2 I_3 \cdots$.

Example: Convert the binary number 0.101001 into a ternary number. Express the ternary number to five places if the conversion is not exact within this range.

1. 3 is represented by 11 in binary
2. $0.101001 \times 11 = 1.111011$ $I_1=1$, $F_1=0.111011$

3. $0.111011 \times 11 = 10.110001 \quad I_2=10,\ F_2=0.110001$

4. $0.110001 \times 11 = 10.010011 \quad I_3=10,\ F_3=0.010011$

   $0.010011 \times 11 = 0.111001 \quad I_4= 0,\ F_4=0.111001$

   $0.111001 \times 11 = 10.101011 \quad I_5=10,\ F_5=0.101011$

5. Since $b_2 > b_1$, conversions to base 3 equivalents are made on the I's.

   I_1: $1 \longrightarrow 1$

   I_2: $10 \longrightarrow 2$

   I_3: $10 \longrightarrow 2$

   I_4: $0 \longrightarrow 0$

   I_5: $10 \longrightarrow 2$

6. The "new number" in ternary form is

$$N_3 = 0.I_1I_2I_3I_4I_5 = 0.12202$$

The conversion is not exact in this case.

C. Ternary Arithmetic

Ternary arithmetic is similar to binary arithmetic or to decimal arithmetic. The following tables define the addition, subtraction, and multiplication of ternary digits.

Definition 2.3. Ternary addition:

Table II. Ternary Addition
(Entries define the results of A + B)

B \ A	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

sum

B \ A	0	1	2
0	0	0	0
1	0	0	1
2	0	1	1

carry

Definition 2.4. Ternary subtraction:

Table III. Ternary Subtraction
(Entries define the results of A - B)

B \ A	0	1	2
0	0	1	2
1	2	0	1
2	1	2	0

difference

B \ A	0	1	2
0	0	0	0
1	1	0	0
2	1	1	0

borrow

Definition 2.5. Ternary multiplication:

Table IV. Ternary Multiplication
(Entries define the results of A ∘ B)

B \ A	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

product

B \ A	0	1	2
0	0	0	0
1	0	0	0
2	0	0	1

carry

The arithmetic operations can easily be performed by following the above definitions.

III. TERNARY ALGEBRA

A. General Description

The mathematical model for a ternary switching circuit is the three-valued propositional calculus or composition algebra. The form used here was first introduced by E. L. Post in his "Introduction to General Theory of Elementary Propositions" (11). Various people have made contributions in this area (6), (10), (17), and the resulting algebra is often referred to as "Post Algebra."

The truth table used in two-valued switching theory can be readily extended to the three-valued case. All possible combinations of input variables are listed at the left and the corresponding value of the function is written at the right. In a ternary system, the truth table for n-input variables has 3^n different combinations of the values of the input variables (i.e., 3^n rows). The function can have any one of the three different values for each input combination. Hence there are 3^{3^n} functions of n-variables in a ternary system. The 27 (3^{3^1}) functions of one input variable are shown in Table V.

Table V. Ternary Functions of One Ternary Variable.

X	f_o	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}
0	0	0	0	0	0	0	0	0	0	1	1	1
1	0	0	0	1	1	1	2	2	2	0	0	0
2	0	1	2	0	1	2	0	1	2	0	1	2

f_{12}	f_{13}	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}	f_{19}	f_{20}
1	1	1	1	1	1	2	2	2
1	1	1	2	2	2	0	0	0
0	1	2	0	1	2	0	1	2

f_{21}	f_{22}	f_{23}	f_{24}	f_{25}	f_{26}
2	2	2	2	2	2
1	1	1	2	2	2
0	1	2	0	1	2

Ternary algebra is developed in three steps. The first step is to define symbols and operations. The second step is to select a set of postulates which are the basis for the algebra. The third step is to formulate some fundamental theorems. These theorems together with the postulates are the working rules for the algebra.

B. Basic Operations

Definition 3.1. Cycling-operation ($'$): The Cycling of x, denoted as x', is defined as

$$x' = (x+1) \bmod 3 \quad \text{where } x \text{ and } x' \in \{0,1,2\}.$$

This definition is also given in Table VI.

Table VI. Cycling-operation

x	x'
0	1
1	2
2	0

Definition 3.2. And-operation ($\cdot$): X and y, denoted as $x \cdot y$ (or simply xy), is defined as the smaller of the two, i.e.,

$$x \cdot y = \min\{x,y\}.$$

This definition is also given in Table VII.

Table VII. And-operation

x	y	$x \cdot y$
0	0	0
0	1	0
0	2	0
1	0	0
1	1	1
1	2	1
2	0	0
2	1	1
2	2	2

It was shown by Post (11) that these two operations constitute a functionally complete set; that is, all ternary functions can be synthesized by some composition of these two operations. Although the two operations as defined constitute a logically complete set, the resulting algebraic expressions can be very long and complicated, hence circuit realizations corresponding to such expressions can be costly. Fortunately other logical gates can be devised that correspond to additional algebraic operations to be introduced. The set of operations will still be logically complete as long as the Cycling and And operations are included.

Definition 3.3. Or-operation (+): x Or y, denoted as $x+y$, is defined as the larger of the two, i.e.,

$$x+y = \max \{x,y\} .$$

Table VIII gives this definition in truth table form.

Table VIII. Or-operation

x	y	x+y
0	0	0
0	1	1
0	2	2
1	0	1
1	1	1
1	2	2
2	0	2
2	1	2
2	2	2

The set $\{0,1,2\}$ and the two binary operations And and Or form a distributive lattice with zero element 0 and universal element 2.

Definition 3.4. Negation- (complementation-) operation $(\bar{\ })$: the negation of x, denoted as $\bar{x}$, is defined as

$$\bar{x} = 2 - x.$$

The truth table of this definition is given in Table IX.

Table IX. Negation-operation

x	$\bar{x}$
0	2
1	1
2	0

Additional operations have been used in the literatures. Some of the more important ones are given in the Appendix, together with theorems associated with them.

C. Postulates

Definition 3.5. An algebraic structure $f = \langle T, 0, 1, 2, \cdot, +, ', = \rangle$, where T is a set, $0, 1, 2 \in T$, $\cdot$ and $+$ are binary operations on T, $'$ is a unary operation on T, and $=$ is the equality relation on T, is called a ternary algebra if and only if it satisfies the following postulates. For all $a, b, c \in T$

P1.	Idempotent	$a+a=a$	$a \cdot a=a$
P2.	Commutative	$a+b=b+a$	$a \cdot b=b \cdot a$
P3.	Associative	$(a+b)+c=a+(b+c)$	$(a \cdot b) \cdot c=a \cdot (b \cdot c)$
P4.	Distributive	$a+bc=(a+b)(a+c)$	$a(b+c)=ab+ac$
P5.	Identity	$a+0=a$	$a \cdot 2=a$

Note that usually 1 is used as the symbol for the identity operation of And. The symbol 2 is used as the identity element in this paper so that it agrees with Table VII which defines the And operation.

D. Theorems

From the above basic definitions and postulates, the following theorems of ternary algebra can be formulated and proved.

Theorem 3.1. $a''' = a$

Proof: We use finite induction

Let $a=0$, then	$a' = 0' = 1$	Def. 3.1
	$a'' = (a')' = 1' = 2$	"
	$a''' = (a'')' = 2' = 0$	"
Let $a=1$, then	$a' = 1' = 2$	"
	$a'' = (a')' = 2' = 0$	"
	$a''' = (a'')' = 0' = 1$	"
Let $a=2$, then	$a' = 2' = 0$	"
	$a'' = (a')' = 0' = 1$	"
	$a''' = (a'')' = 1' = 2$	"

Theorem 3.2. $a \cdot a' \cdot a'' = 0$, $a + a' + a'' = 2$

Proof:

Let $a=0$, then $a'=1$, $a''=2$	Def. 3.1
Let $a=1$, then $a'=2$, $a''=0$	"
Let $a=2$, then $a'=0$, $a''=1$	"
$0 \cdot 1 \cdot 2 = 0$	Def. 3.2 and P3
$0+1+2=2$	Def. 3.3 and P3

Theorem 3.3. $a \cdot 0 = 0$, $a+2=2$

Proof:

$a \cdot 0 = a \cdot (a \cdot a' \cdot a'')$	Th. 3.2
$= (a \cdot a) \cdot a' \cdot a''$	P 3
$= a \cdot a' \cdot a''$	P 1
$= 0$	Th. 3.2

$a+2 = a+(a+a'+a'')$	Th. 3.2
$= (a+a)+a'+a''$	P 3
$= a+a'+a''$	P 1
$= 2$	Th. 3.2

Theorem 3.4. $a\cdot(a+b)=a$, $a+ab=a$

Proof: $a\cdot(a+b) = a\cdot a+a\cdot b$	P 4
$= a+ab$	P 1
$= a\cdot 2+ab$	P 5
$= a\cdot(2+b)$	P 4
$= a\cdot 2$	Th. 3.3
$= a$	P 5
$a+ab = (a+a)\cdot(a+b)$	P 4
$= a\cdot(a+b)$	P 1
$= (a+0)\cdot(a+b)$	P 5
$= a+0\cdot b$	P 4
$= a+0$	Th. 3.3
$= a$	P 5

Theorem 3.5. $ab+ab'+ab'' = a$

$(a+b)\cdot(a+b')\cdot(a+b'') = a$

Proof: $ab+ab'+ab'' = a(b+b'+b'')$	P 4
$= a\cdot 2$	Th. 3.2
$= a$	P 5

$$
\begin{aligned}
(a+b)\cdot(a+b')\cdot(a+b'') &= a+bb'b'' && \text{P 4}\\
&= a+0 && \text{Th. 3.2}\\
&= a && \text{P 5}
\end{aligned}
$$

Theorem 3.6. $\overline{ab}=\overline{a}+\overline{b}$

$\overline{a+b}=\overline{a}\ \overline{b}$

Proof: By perfect induction

Table X. Proof of Theorem 3.6.

a	b	$a+b$	$\overline{a+b}$	$\overline{a}$	$\overline{b}$	$\overline{a}\ \overline{b}$	$a\cdot b$	$\overline{a\cdot b}$	$\overline{a}+\overline{b}$
0	0	0	2	2	2	2	0	2	2
0	1	1	1	2	1	1	0	2	2
0	2	2	0	2	0	0	0	2	2
1	0	1	1	1	2	1	0	2	2
1	1	1	1	1	1	1	1	1	1
1	2	2	0	1	0	0	1	1	1
2	0	2	0	0	2	0	0	2	2
2	1	2	0	0	1	0	1	1	1
2	2	2	0	0	0	0	2	0	0

Note that the principle of duality also holds in the ternary algebra. The application of these theorems will be illustrated in the next chapter.

IV. MINIMIZATION OF TERNARY FUNCTIONS

A. Definitions

One of the basic problems of logical design is the simplification of a given function. The problem of determining the minimal representation of a function is very complicated and has not been completely solved even for the binary case. Minimization in the ternary case, as in the binary case, will depend greatly on the switching devices available. The usual procedure is to establish one parameter, and then the circuitry is minimized with respect to that parameter. There are many ways to select the parameter of interest and the one used here is represented by the cost function, C.

Definition 4.1. The cost function, C, is defined as the number of variable occurrences plus the number of operations in a given function. All three states of a variable are assumed to be available, i.e., the Cycling operation on individual variables is not included in the cost function.

There are basically three ways to minimize a logic function. These are:

1. The algebraic method,
2. the map method, and
3. the tabular method.

Before these methods are discussed, the following terms will be defined. The definitions used here are analogous to those used for the binary system.

Definition 4.2. Literal -- a variable or its cycling, e.g., a, a', a'', etc.

Definition 4.3. Monomial -- a product of literals, e.g., ab, a'bc'', etc.

Definition 4.4. Elementary monomial -- a monomial which contains all the variables on which the function is defined.

Definition 4.5. Implicant -- there are two types of implicants of a function f of n variables.

1) An h-type implicant, I_h, is a monomial satisfying the condition $1 \cdot I_h(\alpha) \leq f(\alpha)$ for every $\alpha \in T^n$.
2) A g-type implicant, I_g, is a monomial satisfying the condition $I_g(\alpha) \leq f(\alpha)$ for every $\alpha \in T^n$.

Definition 4.6. Essential -- a literal is essential in an implicant if the deletion of that literal will make it a non-implicant.

Definition 4.7. Prime implicant -- there are two types of prime implicant of a function f of n variables.

1) An h-type prime implicant, P_h, is an h-type implicant in which every literal is essential.

2) A g-type prime implicant, P_g, is a g-type implicant in which every literal is essential.

Definition 4.8. Cover -- a function Φ covers a function f if and only if

$$f = 0 \Longrightarrow \Phi = 0$$
$$f = 1 \Longrightarrow \Phi = 1$$
$$f = 2 \Longrightarrow \Phi = 2$$

Definition 4.9. Prime implicant table -- a prime implicant table is a table in which columns are represented by prime implicants and rows represented by elementary monomials of a given function. The entries are filled with check marks where the column covers the row.

Definition 4.10. Minimal function -- a function Φ is said to be minimal if Φ covers f and the cost function is minimal.

Theorem 4.1. Any ternary function can be written in the form

$$f = 1 \cdot h(x_1, \cdots, x_n) + g(x_1, \cdots, x_n)$$

where $h(x_1, \cdots, x_n)$ represents the subfunction whose output is 1 and $g(x_1, \cdots, x_n)$ represents the subfunction whose output is 2.

A function in the form of Theorem 4.1 is called in canonical form. Note that the h-subfunction and the g-subfunction cannot contain the same terms.

Example: Give an algebraic expression for the function defined by Table XI.

Table XI. An Arbitrary Function to be Expanded in Canonical Form.

x	y	f
0	0	0
0	1	0
0	2	2
1	0	1
1	1	2
1	2	2
2	0	1
2	1	1
2	2	2

Notation: If w is a variable, let

w represent w = 0

w' represent w = 1

w'' represent w = 2

then, for this example

$$h(x,y) = x'y+x''y+x''y'$$

$$g(x,y) = xy''+x'y'+x'y''+x''y''$$

$$f = 1\cdot h+g = 1\cdot[x'y+x''y+x''y']+xy''+x'y'+x'y''+x''y''$$

Example: Write the logic equations for a sequential 10-counter.

The truth table of a possible 10-counter is given below.

Table XII. Truth Table for a 10-Counter.

p_3	p_2	p_1	P_3	P_2	P_1
0	0	0	0	0	1
0	0	1	0	0	2
0	0	2	0	1	0
0	1	0	0	1	1
0	1	1	0	1	2
0	1	2	0	2	0
0	2	0	0	2	1
0	2	1	0	2	2
0	2	2	1	0	0
1	0	0	0	0	0

The logic equations can be written as

$$P_1 = 1\cdot\{p_3p_2p_1+p_3p_2'p_1+p_3p_2''p_1\}+p_3p_2p_1'+p_3p_2'p_1'+p_3p_2''p_1'$$

$$P_2 = 1\cdot\{p_3p_2p_1''+p_3p_2'p_1+p_3p_2'p_1'\}+p_3p_2'p_1''+p_3p_2''p_1+p_3p_2''p_1'$$

$$P_3 = 1\cdot\{p_3p_2''p_1''\}$$

B. Algebraic Method

The algebraic method of simplification is performed by utilizing the properties of the ternary algebra stated in Chapter III. Since the g-subfunction represents the input combinations that produce the output value 2 and the h-subfunction represents the input combinations that produce the output value 1, the complete function is represented by $f = 1 \cdot h + g$. By the nature of "$\cdot$" and "+" operations, the terms in the g-subfunction can be used as "don't care" terms for the simplification of h-subfunction but not vice versa.

In the binary logic, a very useful relationship for minimization is

$$AB + A\overline{B} = A.$$

The counter part of this relationship in the ternary logic is

$$AB + AB' + AB'' = A.$$

Several examples are given below to illustrate the algebraic method of simplification.

Example: Simplify the function obtained in Section A, page 24.

$$f = 1 \cdot (x'y + x''y + x''y') + xy'' + x'y' + x'y'' + x''y''$$

Each subfunction will be minimized individually.

$$
\begin{aligned}
1\cdot h &= 1\cdot[x'y+x''y+x''y'] \\
&= 1\cdot[(x'y+x''y+x''y')+\overbrace{xy''+x'y'+x'y''+x''y''}^{\text{don't care (from g)}}] \\
&= 1\cdot[x'(y+y'+y'')+x''(y+y'+y'')] \\
&= 1\cdot(x'+x'')
\end{aligned}
$$

$$
\begin{aligned}
g &= xy''+x'y'+x'y''+x''y'' \\
&= y''(x+x'+x'')+x'y' \\
&= y''+x'y'
\end{aligned}
$$

$$f = 1\cdot h+g = 1\cdot(x'+x'')+y''+x'y'.$$

Example: Simplify the logic equations of the 10-counter in Section A.

$$
\begin{aligned}
P_1 &= 1\cdot[p_3p_2p_1+p_3p_2'p_1+p_3p_2''p_1]+p_3p_2p_1'+p_3p_2'p_1'+p_3p_2''p_1' \\
&= 1\cdot[p_3p_1(p_2+p_2'+p_2'')]+p_3p_1'(p_2+p_2'+p_2'') \\
&= 1\cdot(p_3p_1)+p_3p_1'
\end{aligned}
$$

$$
\begin{aligned}
P_2 &= 1\cdot[p_3p_2p_1''+p_3p_2'p_1+p_3p_2'p_1']+p_3p_2'p_1''+ \\
&\qquad p_3p_2''p_1+p_3p_2''p_1' \\
&= 1\cdot[p_3p_2p_1''+p_3p_2'p_1+p_3p_2'p_1'+p_3p_2'p_1'']+p_3p_2'p_1''+ \\
&\qquad p_3p_2''p_1+p_3p_2''p_1' \\
&= 1\cdot[p_3p_2p_1''+p_3p_2'(p_1+p_1'+p_1'')]+p_3[p_2'p_1''+p_2''p_1+p_2''p_1'] \\
&= 1\cdot[p_3p_2p_1''+p_3p_2']+p_3[p_2'p_1''+p_2''(p_1+p_1')] \\
&= 1\cdot[p_3(p_2p_1''+p_2')] \;+p_3[p_2'p_1''+p_2''(p_1+p_1')]
\end{aligned}
$$

$$P_3 = 1\cdot(p_3p_2''p_1'')$$

Example: Simplify the following functions as much as possible.

(a) $f = 1\cdot\{x\} + (x+y+z)(x'+y+z)$

(b) $f = 1\cdot[xy'+xy''+xyz+xz'+xy'z]+x+x'+x'z$

(c) $f = 1\cdot[x(x+y)+x'(x'+z)+x''(x''+w)]+x(y+x'x'')$

(a) $f = 1\cdot\{x\} + (x+y+z)(x'+y+z)$

$$g = (x+y+z)(x'+y+z)$$
$$= xx'+xy+xz+x'y+y+yz+x'z+yz+z$$
$$= xx'+y+z$$
$$1\cdot h = 1\cdot\{x\}$$
$$f = 1\cdot h+g = 1\cdot\{x\} + xx' + y + z$$

(b) $f = 1\cdot[xy'+xy''+xyz+xz'+xy'z] + x+x'+x'z$

$$g = x+x'+x'z$$
$$= x+x'$$
$$1\cdot h = 1\cdot[xy'+xy''+xyz+xz'+xy'z]$$
$$= 1\cdot[(xy'+xy''+xyz+xz'+xy'z)+x]$$
$$= 1\cdot\{x\}$$
$$f = 1\cdot h+g = 1\cdot\{x\}+x+x'$$

(c) $f = 1\cdot[x(x+y)+x'(x'+z)+x''(x''+w)] + x(y+x'x'')$

$$g = x(y+x'x'')$$
$$= xy+xx'x''$$
$$= xy$$

$$
\begin{aligned}
1\cdot h &= 1\cdot[x\cdot(x+y)+x'\cdot(x'+z)+x''\cdot(x''+w)] \\
&= 1\cdot[x+x'+x''] \\
&= 1\cdot 2 \\
&= 1 \\
f &= 1\cdot h+g = 1+xy
\end{aligned}
$$

As in the binary system, the algebraic manipulation may be difficult for a large number of variables or terms, and one is not certain that the function so obtained is minimal.

C. Map Method

Although the map method is easy and convenient, its applicability to the ternary system is very limited due to the large number of combinations. In the binary system, the map method is efficient up to about five or six variables. In the ternary system, the limit is three.

A two-variable map is given in Figure 1. The minimization procedures are similar to those used in binary minimization. These procedures consist of joining corresponding cells into blocks as large as possible.

			x	
		0	1	2
	0	1	4	7
y	1	2	5	8
	2	3	6	9

Figure 1. A two-variable map.

The rules for map method of simplification are:

1. Joining of adjacent cells with equal outputs. The required conditions are 3x1, 3x2, 3x3 arrays.
2. Multiple use of cells for different blocks is permitted.
3. Cells with output value 2 can be considered as "don't care" cells for the formation of blocks with output value 1.
4. Always block the largest possible blocks first.

Example: Use map method to simplify the function given by Table XI, Section A.

The table is converted into map form as shown in Figure 2.

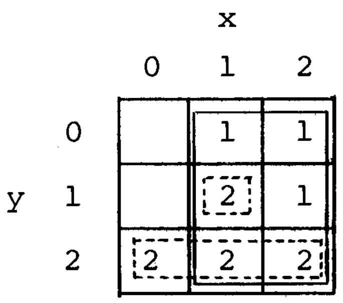


Figure 2. Ternary map for the function given in Table XI.

1. The cells in third row have the same output value, that is, 2. They are combined as one block since they satisfy the condition 1x3. This block is represented by y" .
2. The center cell (cell 5) cannot be combined with its adjacent cells, hence no simplification can be made here. It is represented by x'y'.
3. For the cells 4 through 9, the three cells with output value of 2 can be used as "don't care" cells for the other three cells according to rule 3. Hence a block containing 3x2 cells can be formed. The term that represents this block is $1\cdot(x'+x'')$.

The function that results from this simplification technique is therefore

$$f = 1\cdot(x'+x'') + y'' + x'y',$$

which agrees with what was obtained by algebraic manipulation in Section B, page 27.

For a three-variable function, a three-dimensional map can be used. But with pencil and paper, it is more convenient to use a two-dimensional map as shown in Figure 3. In this map, cell 14 is considered to be adjacent to cells 11, 13, 15, 17 and also cells 5 and 23.

					x					
		0	0	0	1	1	1	2	2	2
	0	1	4	7	10	13	16	19	22	25
z	1	2	5	8	11	14	17	20	23	26
	2	3	6	9	12	15	18	21	24	27
		0	1	2	0	1	2	0	1	2
					y					

Figure 3. A three-variable map.

Cell 11 is adjacent to cells 10, 12, 14 and cells 2 and 20, but not to cell 8. It is probably easier to visualize the adjacencies if the x-variable is displayed in a third dimension as shown in Figure 4. The front-back adjacencies are then clear.

The rules for combination of cells are the same as given for the case of two variables except that now the front-back adjacencies should also be considered. This will complicate the determination of the minimal set of implicants to a great extent.

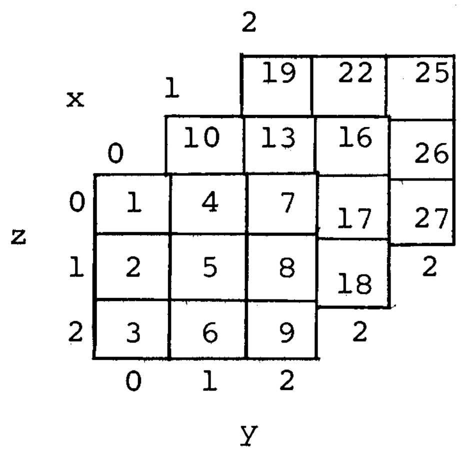


Figure 4. Three-dimensional representation of a three-variable map.

Example: Simplify the function given in Table XIII (d stands for "don't care").

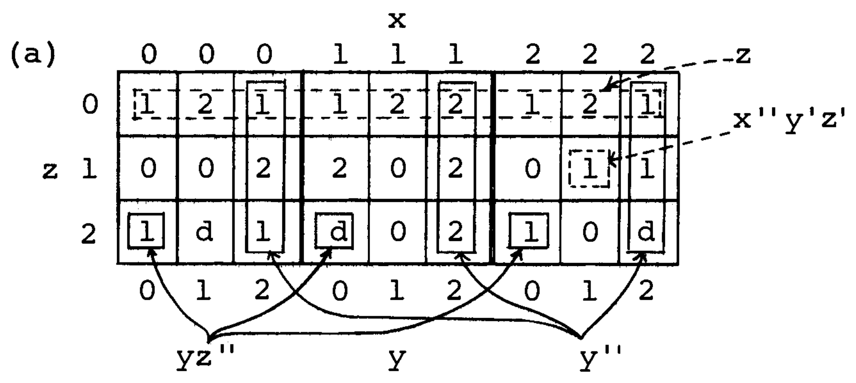
Table XIII. A Three-variable Function

x	y	z	f
0	0	0	1
0	0	1	0
0	0	2	1
0	1	0	2
0	1	1	0
0	1	2	d
0	2	0	1
0	2	1	2
0	2	2	1
1	0	0	1
1	0	1	2
1	0	2	d
1	1	0	2
1	1	1	0
1	1	2	0
1	2	0	2
1	2	1	2
1	2	2	2
2	0	0	1
2	0	1	0
2	0	2	1
2	1	0	2
2	1	1	1
2	1	2	0
2	2	0	1
2	2	1	1
2	2	2	d

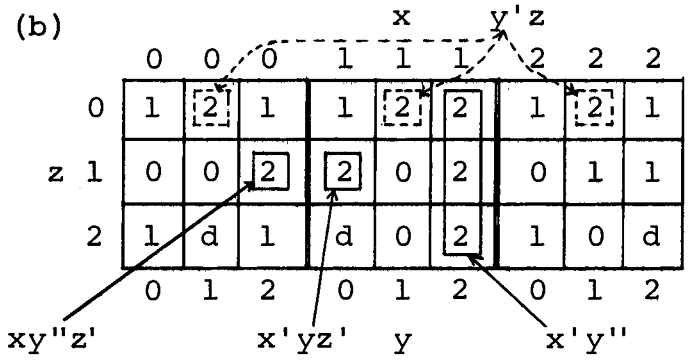
The map that corresponds to this given function is shown in Figure 5. For clarity, the simplifications of the h-subfunction and that of the g-subfunction are shown in Figure 6 (a) and (b) respectively.

	x	0	0	0	1	1	1	2	2	2
	y	0	1	2	0	1	2	0	1	2
z	0	1	2	1	1	2	2	1	2	1
z	1	0	0	2	2	0	2	0	1	1
z	2	1	d	1	d	0	2	1	0	d

Figure 5. Map for the function given in Table XIII.



$1 \cdot h = 1 \cdot [y'' + z + yz'' + x''y'z']$



$g = x'y'' + y'z + xy''z' + x'yz'$

Figure 6. The simplification of
(a) h-subfunction
(b) g-subfunction

From these maps, it is determined that

$$f = 1\cdot h+g = 1\cdot[z+y''+yz''+x''y'z'] + x'y''+y'z+xy''z'+x'yz'.$$

It is obvious that the map method is also very limited, and is not as straightforward as that of the binary system.

D. Tabular Method

For a greater number of variables than about three, neither algebraic nor map methods are practical. However, the tabular method described in this section may be used in these cases. This method can also be programmed for automatic machine simplification.

The procedures for tabular simplification are described by the following algorithm. The algorithm provides a systematic method of applying a limited set of reduction rules in a converging process in order to obtain a minimal function.

Algorithm C:

1. List all elementary monomials corresponding to $f=1$ as group 1, to $f=2$ as group 2, and to the don't care as group 3.
2. Minimize each group using the relationship

   $$AB+AB'+AB'' = A.$$

   Note that groups 2 and 3 can both be used as don't care terms to minimize group 1, and group 3 alone as don't care terms for the minimization of group 2.

3. Check those monomials which have been used to form new monomials. Also, monomials can be repeatedly used if so applicable. Note that if a monomial in group 2 is used as a don't care term for group 1, no check mark is made for the monomial in group 2, as it must be accounted for in group 2.
4. The process continues until no further simplification can be made. All unchecked monomials in groups 1 and 2 are prime implicants.
5. Make prime implicant tables for groups 1 and 2. Choose appropriate prime implicants to form a minimal function.

Example: Simplify the function given in Table XIII using the tabular method.

1. The three groups are listed below in column 1. For ease in tracing the steps of the process, each monomial is assigned an identifying number.
2. The "new monomials" are shown in column 2 together with the identifying number of the "parent monomials."
3. Each monomial used in forming a reduced monomial is checked. No check mark is placed for

the monomial in group 2 if it is used as a don't care for reducing the monomials of group 1.

4. The successively generated "new monomials" are given in successive columns. Note that the "parent monomials" for each "new monomial" are contained in the column immediately preceding the new column.

Table XIV. Minimization of the Three-variable Function of Table XIII.

Group	Column 1	Column 2	Column 3
1	1. 000 ✓	1. (1,3,11) 0X0 ✓	1. (1,9,10)* XX0
	2. 002 ✓	2. (1,5,6) X00 ✓	2. (6,7,12)** X2X
	3. 020 ✓	3. (2,4,19) 0X2	
	4. 022 ✓	4. (2,7,20) X02	
	5. 100 ✓	5. (3,4,12) 02X ✓	
	6. 200 ✓	6. (3,9,15) X20 ✓	
	7. 202 ✓	7. (4,17,21) X22 ✓	
	8. 211	8. (5,13,20) 10X	
	9. 220 ✓	9. (5,14,15) 1X0 ✓	
	10. 221 ✓	10. (6,9,18) 2X0 ✓	
		11. (9,10,21) 22X ✓	
		12. (10,12,16) X21 ✓	
2	11. 010 ✓	13. (11,14,18) X10	
	12. 021	14. (15,16,17) 12X	
	13. 101		
	14. 110 ✓		
	15. 120 ✓		
	16. 121 ✓		
	17. 122 ✓		
	18. 210 ✓		
3	19. 012		
	20. 102		
	21. 222		

* also from (2,6,13)
** also from (5,11,14)

5. The unchecked monomials that remain after the process terminates are the desired prime implicants. They are, in this case:

h-type prime implicants	211
	0X2
	X02
	10X
	XX0
	X2X
g-type prime implicants	021
	101
	X10
	12X

6. Prime implicant tables for selection of appropriate prime implicants.

Table XV. h-Type Prime Implicant Table.

	211	0X2	X02	10X	XX0	X2X
000					✓	
002		✓	✓			
020					✓	✓
022		✓				✓
100				✓	✓	
200					✓	
202			✓			
211	✓					
220					✓	✓
221						✓

Table XVI. g-type Prime Implicant Table.

	021	101	X10	12X
010			✓	
021	✓			
101		✓		
110			✓	
120				✓
121				✓
122				✓
210			✓	

The most economical function from Table XV is the combination of X2X, XX0, X02, 211, and that from Table XVI is the combination of 12X, X10, 101, 021 (all prime implicants in this case). The function can be written as

$$f = 1\cdot[y''+z+yz''+x''y'z']+x'y''+y'z+x'yz'+xy''z'$$

which agrees with that obtained in Section C, page 36.

V. TERNARY SWITCHING CIRCUITS AND STORAGE DEVICES

If the advantages of the ternary logic are to be realized, it is necessary to use a simple, inexpensive ternary element. Most of the electromagnetic and electronic switching elements are, however, two-valued devices. There is, at present, no device in common use that inherently has three stable states.

The nearest approximation to this requirement is the non-linear Hall-effect ternary logic element (4) and the thin magnetic film for storage in (17). A magnetic ternary device was developed by Anderson and Dietmeyer (1). But none of these devices are well-developed, and each is either expensive, or not practical for other reasons.

Ternary switching circuits could also be built up with devices of a basically binary type. Hallworth and Heath (2) have devised several circuits using p-n-p and n-p-n transistor pairs to fulfill the purpose. A set of tunnel-diode circuits for ternary logic have been developed by the faculties of Engineering in Osaka University, Japan (3). The tunnel-diodes are fast but expensive, and always present the problem of synchronization. The circuits presented in this chapter may not be the best but they will do the job. More development effort is needed to make them more suitable.

A. Basic Gates

The basic gates of ternary logic together with the symbols that represent each gate are shown in Figures 7, 8, 9 and 10. The diode-transistor scheme will be employed as other ternary devices are not practically usable yet. The representations of logic levels by voltages are given below.

logic 0	$V \geq +2v$	(typical value 2v)
logic 1	$-1v \leq V \leq +1v$	(" " 0v)
logic 2	$V \leq -2v$	(" " -2v)

Note that the ternary Or-gate and And-gate (Figures 7 and 8) are the same as binary Or-gate and And-gate realizations, hence no explanation will be given here.

In Figure 9, if the input represents "0" (+2v), the base of T_2 is near +2v. R_1, R_2, and R_3 are designed so that T_2 is saturated, and T_3 keeps the output proportionately as far below the emitter of T_3 as point X is above it. This gives an output of approximately 0v, or the "1" state. If the input is "1" (0v), the base of T_2 is near ground potential, and R_1, R_2 and R_3 still keep T_2 saturated. Thus the output is kept proportionately below the emitter of T_3 by T_3. This gives an output of -2v, which represents the truth value "2". If the input is "2" (-2v), T_2 is cut off. R_1, R_2 and R_6

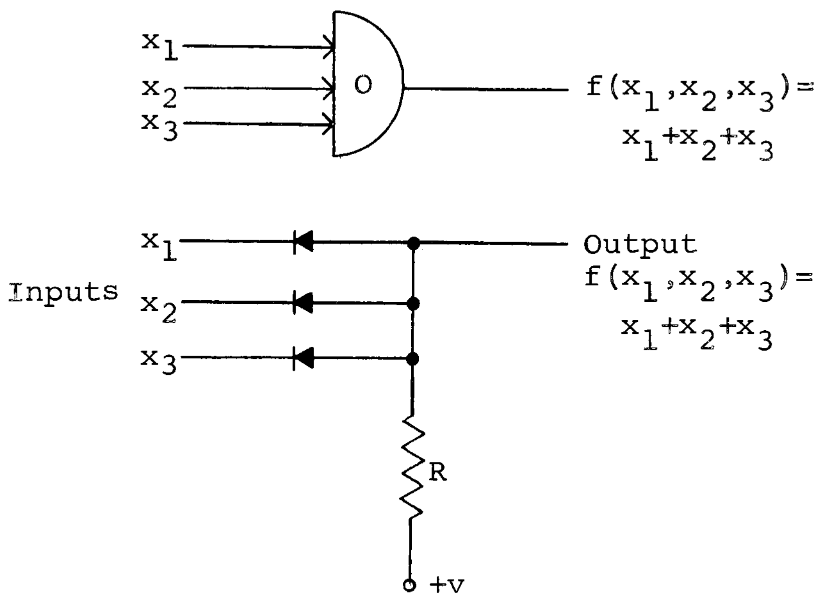


Figure 7. Or-gate

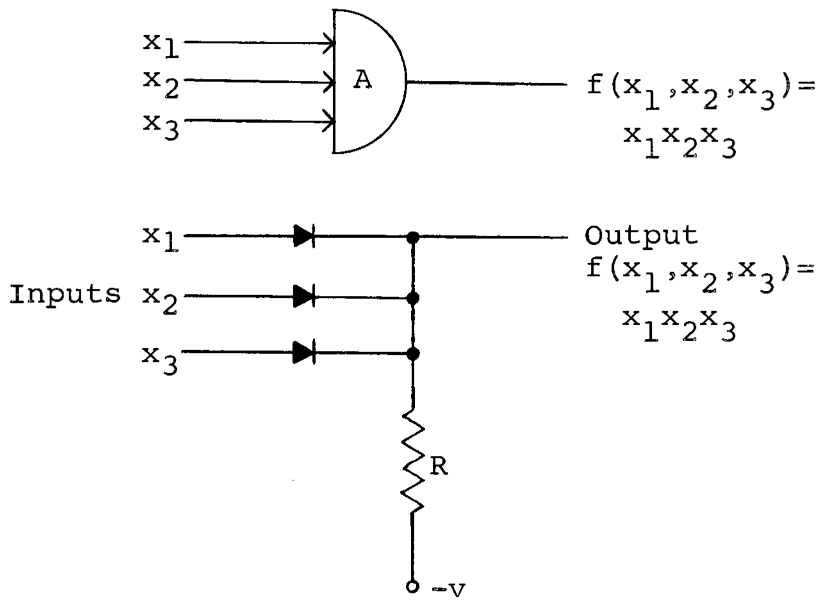
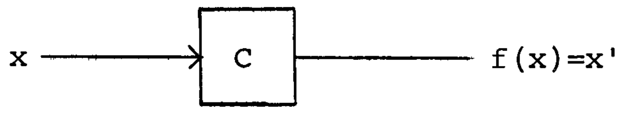


Figure 8. And-gate



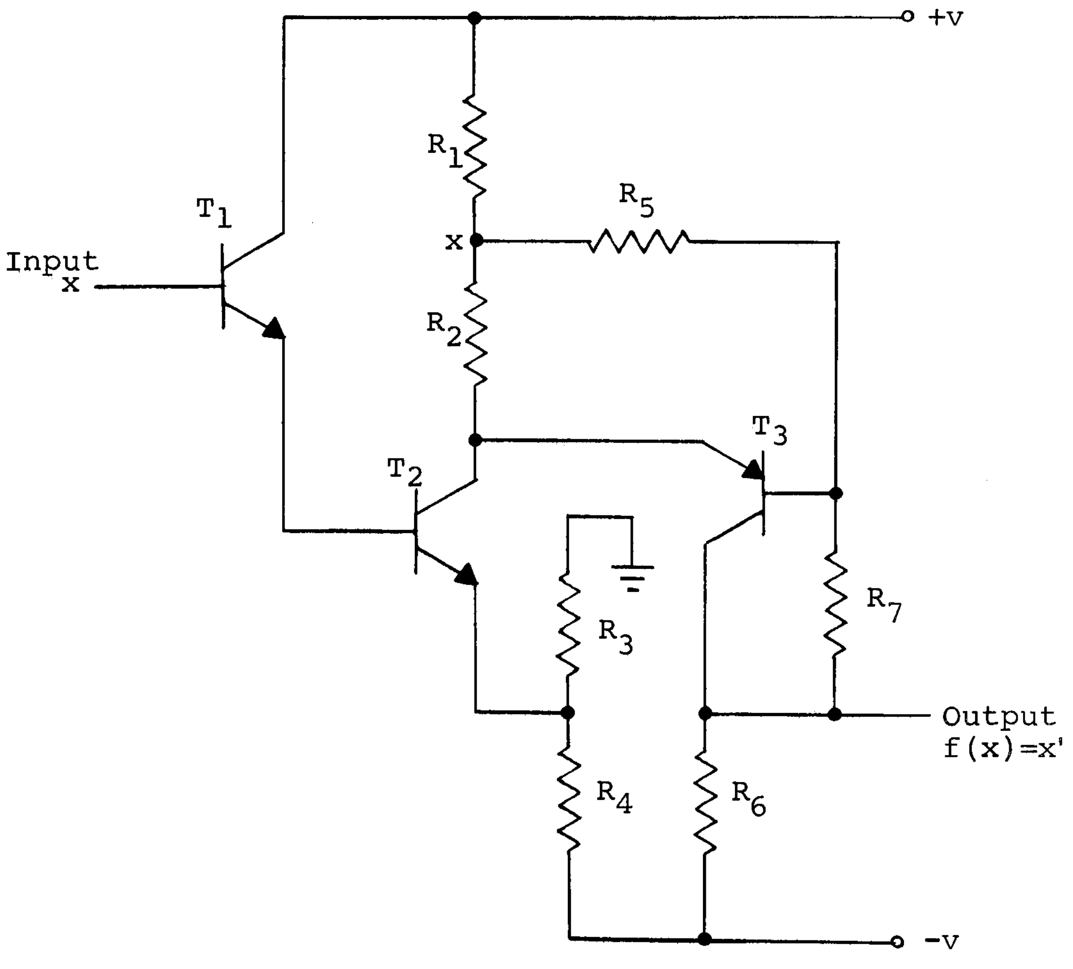
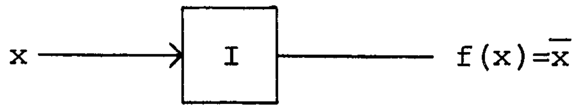


Figure 9. Cycling-gate.



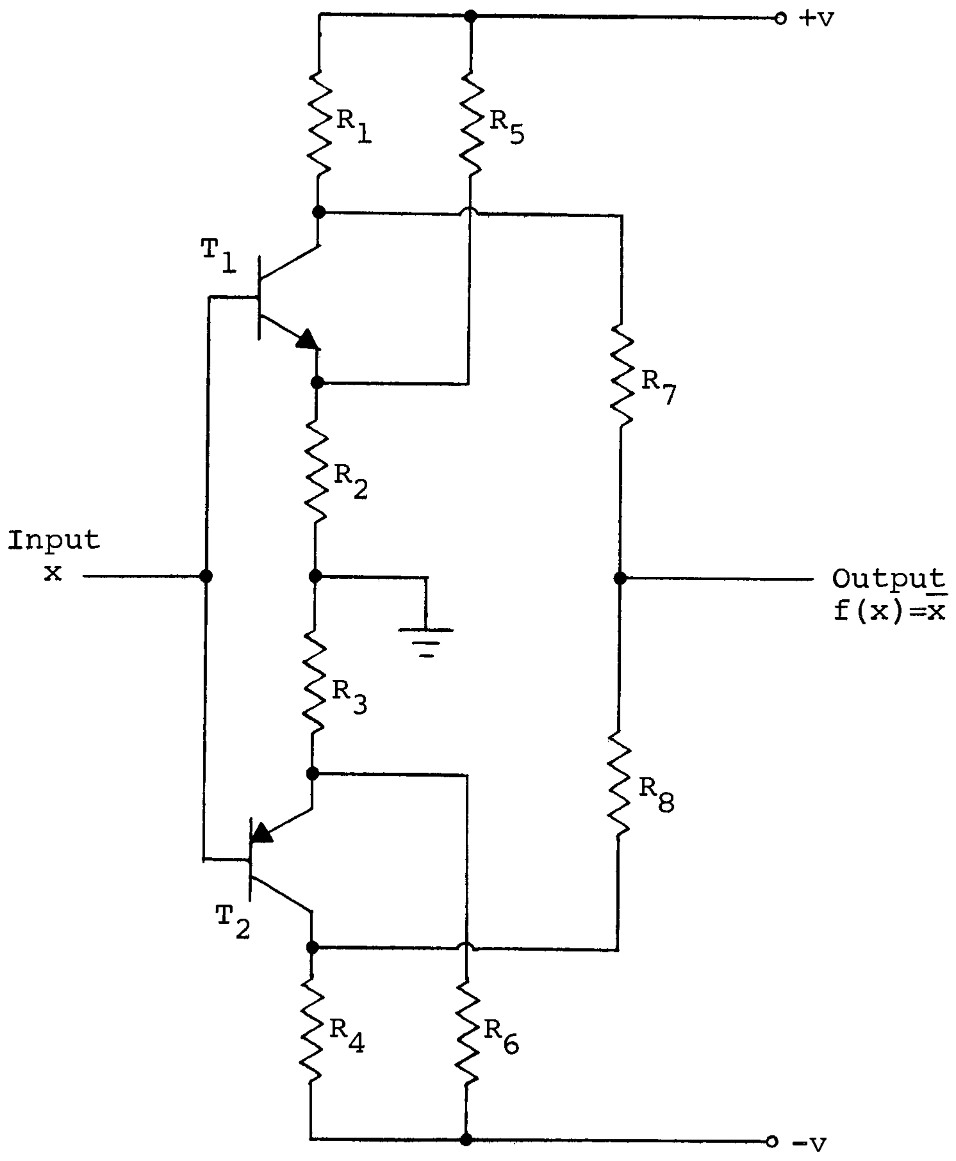


Figure 10. Inverter

together with the circuit of T_3 now give an output of +2v, which is adequate for the truth value "0".

In Figure 10, if the input is positive (+2v), T_1 acts as an amplifier and is saturated while T_2 is cut off. The output is then determined by R_4, R_7 and R_8 and the output is negative. If the input is at zero voltage, both transistors are cut off, and the output is approximately zero volts, because of the symmetry of the circuit. If the input is negative (-2v), T_2 is saturated while T_1 is cut off. The output is determined by R_1, R_7 and R_8 and is positive.

B. Memory Elements

The work-horse of the binary digital system is the bistable circuit or flip-flop. The ternary versions of such a circuit will be referred to as tristable circuit or ternary-flip-flop or simply TFF. Several circuits are proposed below. They are designed by making analogy with binary circuits.

A R-S TFF is shown in Figure 11.

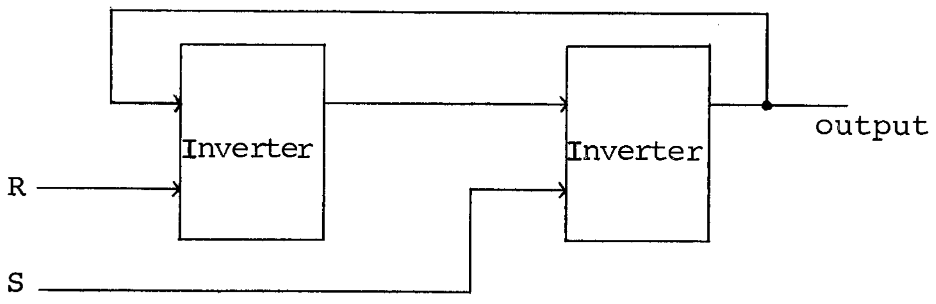


Figure 11. R-S TFF composed of two inverters.

Table XVII defines the operation of R-S TFF.

Table XVII. Operation of R-S TFF

R		S	Output
0	or	2	0
1	and	1	1
2	or	0	2

Note the similarity between this circuit and the binary R-S FF in Figure 12 which is composed of two NOR-gates.

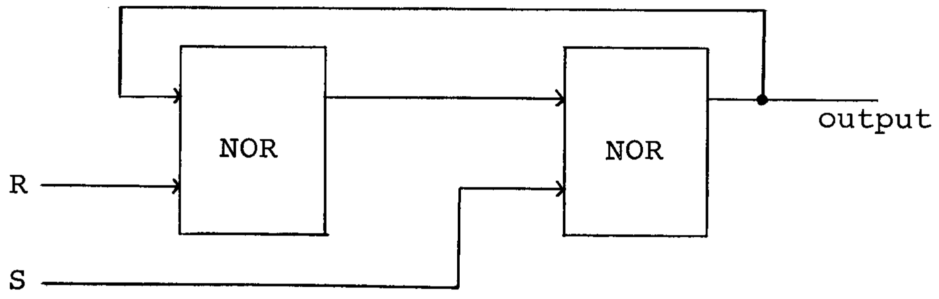


Figure 12. R-S FF in binary system.

While the R-S TFF is not very useful in its present form, the R-S-T TFF shown in Figure 13 should be of great use. The circuit is composed of three Cycling-gates. A triggering voltage of positive polarity applied to R will produce the following logical outputs:

r : 1

s : 2

t : 0

If the triggering is applied to S, the outputs are

r : 0

s : 1

t : 2

If the triggering is applied to T, the outputs are

r : 2

s : 0

t : 1

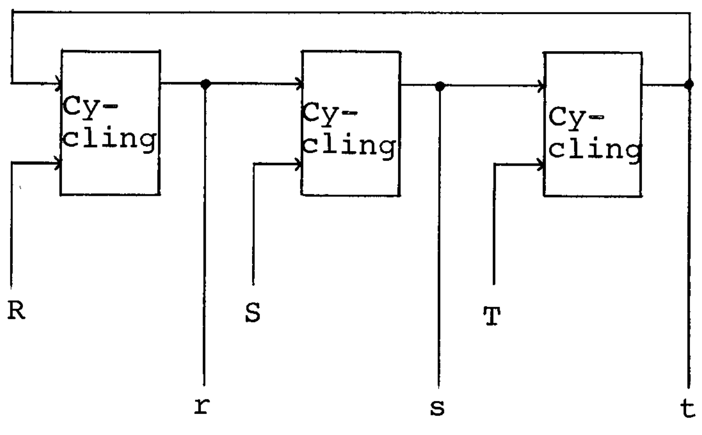


Figure 13. R-S-T TFF.

An astable TFF is shown in Figure 14.

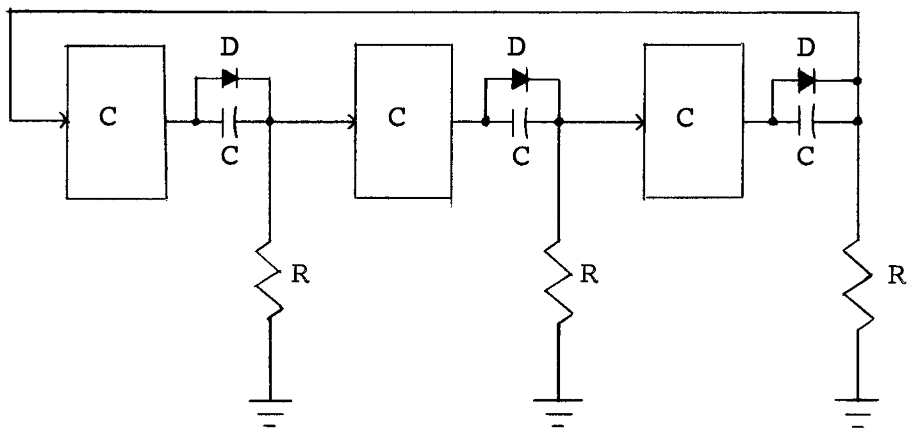


Figure 14. Astable TFF.

A monostable TFF can be constructed by simply short-circuiting one of the coupling circuits in Figure 14.

C. Ternary Memory

The greatest drawback of the ternary digital system is the unavailability of economical storage devices. Several authors have tried to develop a ternary storage element but all failed under the criteria of practicality, simplicity and economy. Figure 15 shows one method of accomplishing this task. It is composed of two two-state magnetic cores to provide one ternary digit of storage.

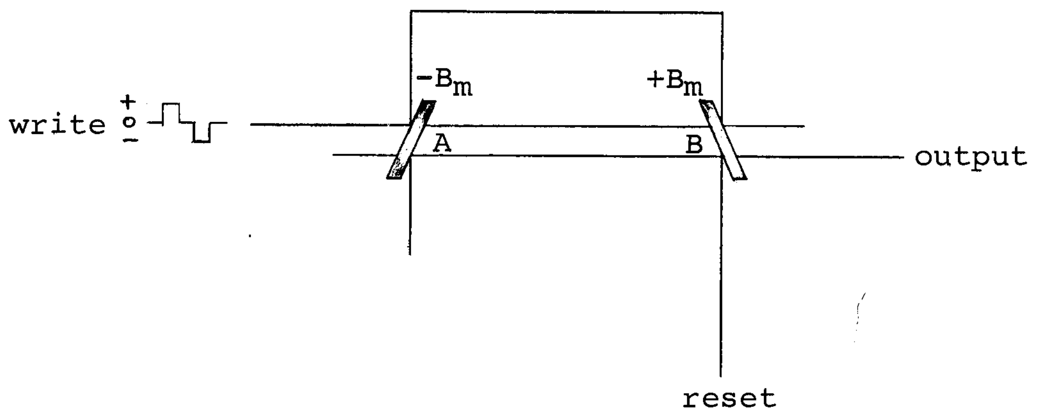


Figure 15. Ternary core switch.

The cores are wired in series as shown. Initially, one core is in $+B_m$ state while the second core is in the $-B_m$ state. The input pulses to write the value "2" are of negative polarity, and thus can only switch core B. The input pulses to write the value "0" are of positive polarity and can only switch core A. The zero is represented by no pulse, and the core states are not changed in this case. The reset pulse which occurs subsequent to the write pulse is driven through both cores in such a way that it will return either of the cores back to its original state. An output pulse of positive or negative polarity or no pulse is obtained on the output winding (when the cores are reset) to indicate if a "0", "2", or "1" was previously written. The magnitude of the write and reset currents in both cores will have to be of sufficient amplitude to switch the cores.

A storage unit of the "coincident current" type is shown in Figure 16. This circuit can be used in the matrix form shown in Figure 17. The A write pulse will be of positive or negative polarity according to the digit which is to be stored. The B write pulses are

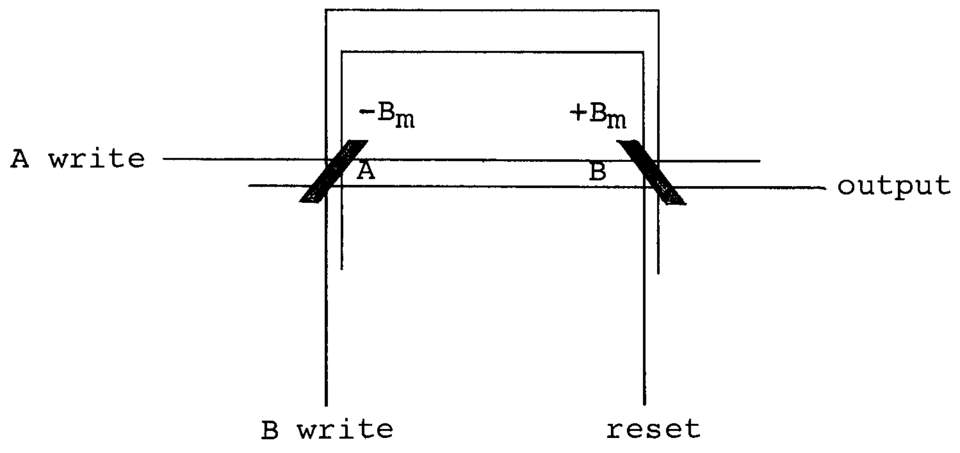


Figure 16. Coincident current core storage unit.

always of positive polarity. All write current pulses have an amplitude of $I_m/2$, thus only one chosen core will get the full switching current. The selection of any core in the matrix is effected by selecting a row and a column as in the case of binary coincident current memory.

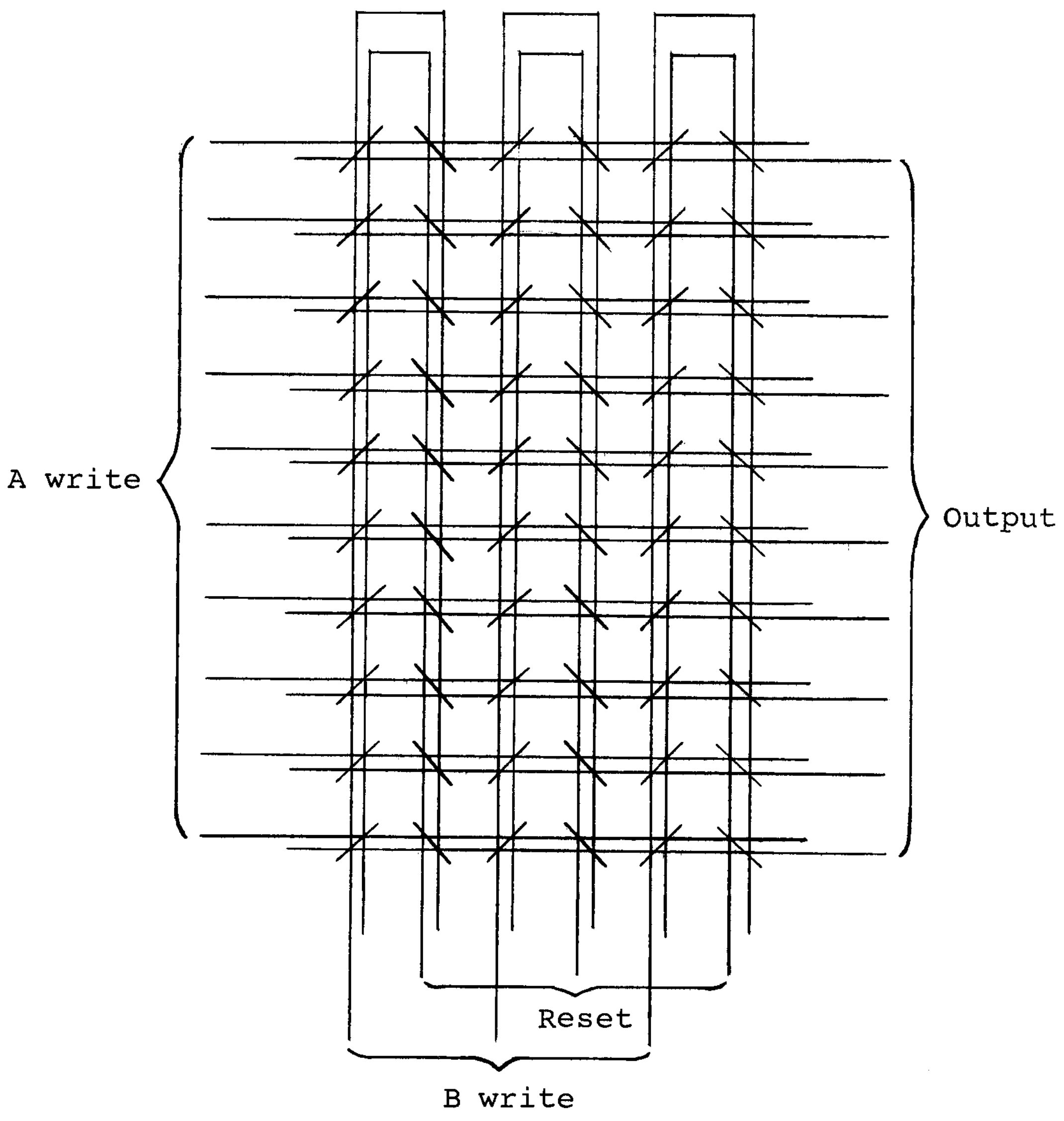


Figure 17. Ternary core memory.

VI. CONCLUSION

In this paper, a systematic method of designing a ternary digital system was described. The system of logic described is characterized by a rather close analogy to the systems of Boolean algebra which are normally employed in the cases where two values only are permitted. The method described here is not the only possible method, but it is one of the simplest and can be easily understood and applied by designers familiar with binary techniques.

Apart from the arithmetic computer applications, the technique provides a very practical method for the realization of special logical functions concerned with three-valued digital system. It is foreseeable that a completely ternary-based computer will find a useful place in a digital control system. The advancing microelectronic techniques may well lead to more economical ternary-based circuits where thin-magnetic film assemblies are integrated with thin-film semiconductor assemblies.

At present, however, binary systems are still preferred. Much work still needs to be done, in both theory and hardware, before ternary systems can be practically realized. Whether ternary systems ultimately will be widely used or not depends mainly on the development of economical ternary switching elements and storage devices. The problems are both challenging and interesting. It is hoped that digital technology will see further advances in this area.

BIBLIOGRAPHY

1. Anderson, D. J. and D. L. Dietmeyer. A magnetic ternary device. IEEE Transactions on Electronic Computers EC-12:911-914. 1963.

2. Hallworth, R. P. and F. G. Heath. Semiconductor circuits for ternary logic. IEE Proceedings 109(C):219-225. 1962.

3. Hasegawa, T. et al. Tunnel-diode circuits for ternary logic. Electronics and Communications in Japan 47:88-95. 1964.

4. Hooi, C. F. and J. L. Weaver. Non-linear Hall-effect ternary logic element. Solid-State Electronics 7:311-321. 1964.

5. Lee, C. Y. and W. H. Chen. Several-valued combinational switching circuits. AIEE Transactions 75:278-283. 1956.

6. Lowenshuss, Oscar. Non-binary switching theory. IRE National Convention Record 6(4):305-317. 1958.

7. Merrill, Roy D. A tabular minimization procedure for ternary switching functions. IEEE Transactions on Electronic Computers EC-15:578-585. 1966.

8. Miller, Raymond E. Switching theory. Vol. 1. New York, John Wiley, 1966. 351p.

9. Morris, D. J. and W. Alexander. An introduction to the ternary code number system. Electronic Engineering 32:554-557. 1960.

10. Muehldorf, E. I. Multivalued switching algebras and their applications in digital systems. National Electronics Conference, Proceedings 15:467-480. 1959.

11. Post, Emil L. Introduction to a general theory of elementary propositions. American Journal of Mathematics 43:163-185. 1921.

12. Pugh, A. Application of binary devices and Boolean algebra to the realization of 3-valued logic circuits. IEE Proceedings 114:335-338. 1967.

13. Salter, Forrest. A ternary memory element using a tunnel diode. IEEE Transactions on Electronic Computers EC-13:155-156. 1964.

14. Santos, J. and H. Arango. Base 3 vs base 2 synchronous arithmetic units. IEEE Transactions on Electronic Computers EC-13:608-609. 1964.

15. Santos, J. and H. Arango. On the analysis and synthesis of 3-valued digital systems. AFIPS Spring Joint Computer Conference, Proceedings 25:463-475. 1964.

16. Vacca, R. A three-valued system of logic and its applications to base three digital circuits. In: International Conference on Information Processing, Proceedings. Paris, UNESCO, 1959. p.407-414.

17. Yoeli, M. and G. Rosenfeld. Logical design of ternary switching circuits. IEEE Transactions on Electronic Computers EC-15:19-29. 1965.

APPENDIX

APPENDIX

Two more operations have been widely discussed in ternary switching theory. These additional operations will, in many instances, reduce the hardware cost, although at the same time they will complicate the algebraic manipulation. The definitions of these two operations are given below.

Definition A.1. J_i-operation is defined as

$$J_i(x) = \begin{cases} 0 \text{ if } i = x \\ 2 \text{ if } i \neq x \end{cases}$$

$$i, x \in \{0,1,2\} .$$

Definition A.2. K_i-operation is defined as

$$K_i(x) = \begin{cases} 0 \text{ if } i \neq x \\ 2 \text{ if } i = x \end{cases}$$

$$i, x \in \{0,1,2\} .$$

Definitions A.1 and A.2 are summarized in Table XVIII.

Table XVIII. J_i-, K_i-operation

X	$J_o(x)$	$J_1(x)$	$J_2(x)$	$K_o(x)$	$K_1(x)$	$K_2(x)$
0	0	2	2	2	0	0
1	2	0	2	0	2	0
2	2	2	0	0	0	2

Note that $K_i(x) = \overline{J_i(x)}$ and $J_i(x) = \overline{K_i(x)}$

The J_i-operation and K_i-operation can be expressed in terms of two fundamental operations, And and Cycling.

$$J_o(x) = (x'\cdot x'')''$$
$$J_1(x) = (x\cdot x')''$$
$$J_2(x) = (x''\cdot x)''$$
$$K_o(x) = (x\cdot x')''\cdot (x\cdot x'')''$$
$$K_1(x) = (x'\cdot x'')''\cdot (x\cdot x'')''$$
$$K_2(x) = (x'\cdot x'')''\cdot (x\cdot x')'' .$$

The relationships between J_i-operation and K_i-operation are:

$$J_o(x) = K_1(x)+K_2(x)$$
$$J_1(x) = K_o(x)+K_2(x)$$
$$J_2(x) = K_o(x)+K_1(x)$$
$$K_o(x) = J_1(x)\cdot J_2(x)$$
$$K_1(x) = J_o(x)\cdot J_2(x)$$
$$K_2(x) = J_o(x)\cdot J_1(x)$$

Also the following relations are useful in reducing algebraic expressions.

$$K_o(x)+K_1(x)+K_2(x) = 2$$

$$J_o(x)\cdot J_1(x)\cdot J_2(x) = 0$$

$$K_i(x)\cdot K_j(x) = \begin{cases} K_i(x) & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

$$J_i(x)+J_j(x) = \begin{cases} J_i(x) & \text{if } i = j \\ 2 & \text{if } i \neq j \end{cases}$$

It is well-known that binary functions can be expanded in disjunctive normal form and conjunctive normal form by applying the expansion theorems (due to Shannon):

$$f(x_1,x_2,\cdots,x_n) = x_1 f(1,x_2,\cdots,x_n)+\overline{x}_1 f(0,x_2,\cdots,x_n)$$

or,

$$f(x_1,x_2,\cdots,x_n) = [x_1+f(0,x_2,\cdots,x_n)] [\overline{x}_1+f(1,x_2,\cdots,x_n)]$$

Not surprisingly, there exists a similar pair of expansion theorems in the ternary system:

$$\begin{aligned} f(x_1,x_2,\cdots,x_n) = {} & J_o(x_1)f(0,x_2,\cdots,x_n)+ \\ & J_1(x_1)f(1,x_2,\cdots,x_n)+ \\ & J_2(x_1)f(2,x_2,\cdots,x_n) \end{aligned}$$

or,

$$\begin{aligned} f(x_1,x_2,\cdots,x_n) = {} & [K_o(x_1)+f(o,x_2,\cdots,x_n)] \\ & [K_1(x_1)+f(1,x_2,\cdots,x_n)] \\ & [K_2(x_1)+f(2,x_2,\cdots,x_n)] \end{aligned}$$

If the cost of a J_i-gate or a K_i-gate is the same as that of an And-gate or an Or-gate, then the use of J_i-operation and K_i-operation will tend to reduce the cost of a given function.

Example: Reduce the following equation (a) without J_i-, K_i-operation, (b) with J_i-, K_i-operation.

$$f = 1\cdot[x+yz+yz''+x''] + y'z'+yz'+y''z+y''z''+y''z'$$

(a)

$$\begin{aligned} g &= y'z'+yz'+y''z+y''z''+y''z' \\ &= z'(y+y')+y''(z+z'+z'') \\ &= z'(y+y')+y'' \end{aligned}$$

$$\begin{aligned} 1\cdot h &= 1\cdot[x+yz+yz''+x''] \qquad \text{don't care (from g)} \\ &= 1\cdot[(x+yz+yz''+x'')+y'z'] \\ &= 1\cdot[x+x''+y(z+z'+z'')] \\ &= 1\cdot[x+x''+y] \end{aligned}$$

$$f = 1\cdot h+g = 1\cdot[x+x''+y]+z'(y+y')+y''$$

cost = 14

(b) $g = z'(y+y')+y''$ from part (a)

$= z'J_2(y)+y''$

$1\cdot h = 1\cdot[x+x''+y]$ from part (a)

$= 1\cdot[J_1(x)+y]$

$f = 1\cdot h+g = 1\cdot[J_1(x)+y] + z'J_2(y)+y''$

cost = 11. A saving of 3.

Example: Simplify the following expression as much as possible.

$f = xx'+xx''+x'x''+x''$

(a) Without J_i-, K_i-operation

$f = xx'+xx''+x'x''+x''$

$= xx'+x''$ cost = 5

(b) With J_i-, K_i-operation

$f = xx'+xx''+x'x''+x''$

$= x(x'+x'')+x''(x'+x'')$

$= (x'+x'')(x+x'')$

$= J_o(x)\cdot J_1(x)$

$= K_2(x)$ cost = 2

There are many operations in ternary algebra which are analogous to the sheffer stroke operation in binary algebra. One of these is the following:

Table XIX. Another Binary Operator for the Ternary System. (Analogous to the binary "stroke" function)

x	y	$x \mid y$
0	0	1
0	1	1
0	2	1
1	0	1
1	1	2
1	2	2
2	0	1
2	1	2
2	2	0

It can be seen that $x' = x \mid x$

$$x \cdot y = (x \mid y)'' .$$

The operation "$\mid$" is therefore logically complete. The simplification achieved, however, would only be one of notation, whereas the actual physical circuits would be more complicated, both because of the lesser flexibility of the operation and because of the fact that a physical circuit realizing the operation would be rather complicated in itself.