A SYNTHESIS TECHNIQUE FOR THE RESOLUTION OF LARGE SCALE FLUID LOGIC NETWORKS

By

JIM BASUKI SURJAATMADJA

Bachelor of Engineering
Institute of Technology of Bandung
Bandung, Indonesia
1970

Master of Engineering
Institute of Technology of Bandung
Bandung, Indonesia
1971

Master of Science
Oklahoma State University
Stillwater, Oklahoma
1972

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
December, 1976

A SYNTHESIS TECHNIQUE FOR THE
RESOLUTION OF LARGE SCALE
FLUID LOGIC NETWORKS

Thesis Approved:

[illegible]
Thesis Adivser

Paul A. McCollum

[illegible]

James E Bose

Norman N Durham
Dean of the Graduate College

997121

ACKNOWLEDGMENTS

The study is concerned with the development of a new synthesis technique for asynchronous sequential logic networks. The primary objective of the research effort is to provide the logic designer with a powerful tool for synthesizing extremely large digital control systems. All steps which have been selected to be used in the synthesis were evaluated extensively; and only steps which are suitable for large scale synthesis were retained.

It is my pleasure to utilize this opportunity to express my appreciation to my major adviser, Professor E. C. Fitch, for his continuous guidance and encouragements throughout this study. I am especially indebted to him for all the time and effort which he has contributed throughout the pursuance of the research effort. Sincere appreciation is also extended to all of my committee members, Professors P. A. McCollum, J. E. Bose, and D. D. Lingelbach for their invaluable assistance during the study and in the preparation of the manuscript.

Special thanks are also extended to the Fluid Power Research Center Staff for their assistance and contributions during the study. Also, appreciation is acknowledged to the Fluid Power Research Center and its sponsoring companies, also to the National Science Foundation for the grant during the period of the research effort.

Finally, special gratitude is expressed to my wife, Agnes, and my parents, for their understanding, encouragement and sacrifices during the pursuance of this study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Fluid power has been utilized for over two centuries. Its popularity has continued to increase since it is one of the most effective means of transmitting high levels of power. As with electricity, fluid power does not utilize bulky linkages; and it may be transmitted through long transmission lines and through tight spaces where the utilization of mechanical linkages would not be permissible.

Progressing through the years, fluid power invaded the area of control in the early part of the 20th century (1), both in digital and analog control. In the area of digital control, fluid elements have been proven capable of performing logical decisions necessary to replace the human operator. The use of fluid elements eliminates the need of costly interfaces when fluids are used as the "muscle power" of the controlled system. In addition, fluid elements are very reliable when operating under adverse conditions; such as heat, vibrations, mechanical impacts, etc. This feature has made the utilization of digital fluid elements (or fluid logic elements) increasingly popular.

At the early stages of digital fluid control, the degree of automation which was attained was very primitive. In designing such systems, both intuition and a rudimentary knowledge of logic elements were utilized. The methods of design which were employed reflected those of an art rather than a science. Progress in the area of fluid logic controls have brought with it the demand for more and more sophisticated design methods. Means of designing circuits employing intuitive processes have been proven to be time consuming and not rewarding, since they reduce to an endless game of avoiding errors while searching for the desired solution.

This study has been directed towards the development of computer oriented algorithms needed for synthesizing large scale, stochastic type sequential fluid logic networks. In addition, applicable combinational synthesis methods are assessed in a unique, generalized approach. The possibilities of utilizing special elements such as NOR elements have been considered.

CHAPTER II

PREVIOUS INVESTIGATIONS

Ever since the early part of the 20th century, the use of sequential networks has served as a substitute for the decision logic of the human operator (2). Initially, intuitive approaches were adequate due to the simplicity of the problems and the fact that the art of switching theory was very limited. This state began to change with the injection of Boolean Algebra to switching theory by C. E. Shannon (3), who utilized the algebra for the representation and manipulation of relay networks. Shannon's approach transformed previous thinking and laid the foundation of modern switching theory and design.

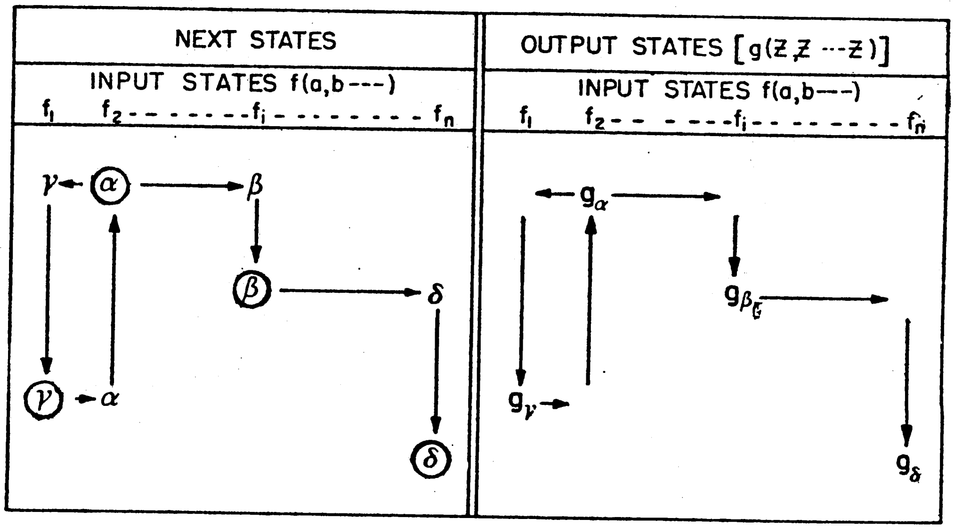
In 1953, M. Karnaugh (4) established another milestone in the history of switching theory. His contribution concerned the introduction of an effective means for simplifying Boolean equations using a map. Karnaugh's map, which is basically an improvement of the Veitch Chart (5), provided a new insight into the problem of network simplification. It was also the basis for the now famous sequential synthesis method developed by D. A. Huffman (6) a year later.

Huffman's sequential logic synthesis method was first formally introduced to the area of fluid logic by E. C.

Fitch (7) in 1964, and is now commonly referred to as the "Classical Technique". The technique evaluates a logic problem using a table which possesses a direct relationship with the Karnaugh map. This table is known as the "Primitive Flow Table" and is illustrated in Table I.

TABLE I

THE PRIMITIVE FLOW TABLE



Each circled entry in the table indicates a stable state and relates to a distinct machine state, while uncircled entries denote unstable states and refer to the next possible machine state that may be attained following a change of the inputs. Each column of the table is reserved for one state of the inputs, while each row relates to the internal states of the machine. These internal states are represented by a group of secondary elements which are called memory elements.

Huffman presented a method for modifying the table such that a near minimal row flow table is achieved, hence minimizing the number of memory elements required. This row minimization technique was reformulated and improved by many scientists in order to obtain the absolute minimal row flow table in a most convenient manner (8, 9, 10, 11, 12, 13, 14, 15). One of the most mechanized schemes was developed by A. Grasselli and F. Luccio (16, 17, 18) and N. Biswas (19). The improved version of the method is capable of producing the minimal row (or merged) flow table in a most efficient manner.

Huffman's method proceeds with the construction of the operational flow table (OFT), followed immediately by the development of the network maps. The need for an OFT actually lies in the potential danger that races may occur between the memories, a situation which must be avoided in order that an undesired state is not attained accidentally.

It is the construction of the "optimal" OFT that has created a major obstacle in the Classical Synthesis Technique. Creating a minimal row OFT constitutes a major problem by itself; since without resorting to pure trial and error approaches one would never be sure that the true minimal row OFT has been obtained. One consolation is that possessing a true minimal OFT does not necessarily guarantee that a minimal-hardware network can be achieved.

The inherent problems in the Classical method and the crucial need for a fully mechanized fluid logic synthesis method capable of producing near minimal networks prompted J. H. Cole (20) to formulate a tabular method for synthesizing feedback type fluid logic networks. Although his method, often referred to as the Change Signal Method, is effective only for a certain class of deterministic sequential circuits, it is capable of obtaining near minimal solutions. In the establishment of this method, one point was clarified; i.e., the minimal hardware network is not necessarily obtained by a minimal memory network.

In 1969, G. E. Maroney (21, 22) developed a concept that used the "total" or complete input state. His Diconesyn III synthesis method was able to handle virtually any type of logic network. Modifications to his method have provided total mechanization (23, 24, 25).

Other attempts to devise better and better techniques for synthesizing fluid logic networks are evident in the literature. The method introduced by R. M. H. Chan and K. Foster (26) must be recognized as the first method that uniquely places all memories at the output. This nondiscriminative memory assignment method often creates major difficulties in attempting to eliminate races that have resulted from this random memory assignment procedure. Such elimination often requires that additional memory elements be utilized; and hence, an unnecessary increase in the network complexity may occur.

An attempt to achieve minimal networks by selective matching of memory functions to the outputs was made by Surjaatmadja (27, 28) in 1973. The approach showed a great promise in typical applications in fluid logic as demonstrated not only by the degree of simplification achieved, but also by the ability of reducing the ill-effects of hazards. The method was successfully followed by another approach, which utilizes a special class of outputs to perform memory functions (29). As a correct classification of the outputs results in the reduction of the required hardware, it is this direction that is pursued by the author in this investigation.

CHAPTER III

STATEMENT OF THE PROBLEM

The purpose of this investigation was to advance computer oriented algorithms for synthesizing large scale, stochastic type sequential fluid logic networks. The approach should enable the establishment of near-minimal circuits in the most efficient manner.

The method includes the development of an input-output circuit selection criterion, the development or selection of appropriate combinational synthesis techniques which forms the foundation for the intended sequential synthesis technique.

In general, the plan of attack involves the study of previously established methods and the possible incorporation of such methods in the development of the new synthesis technique. New algorithms are to be devised whereever necessary. After the formulation, the method is to be demonstrated.

CHAPTER IV

SWITCHING CIRCUIT ALGEBRA

4.1 Evolution of the Algebra

Switching circuit theory is based upon pure logic--an art that is intrinsically possessed by mankind. With it, man has been able to make decisions and perform different tasks. Although there are many attempts throughout the history towards the formulation of logic, no mathematical assessment had been made until G. Boole (30) formulated a unique algebraic representation of logic in 1854. This "new" branch of the algebra has been improved throughout the years and is currently known as "Boolean Algebra".

Ironically, it was not until 1938 that Boolean algebra finally found its place in the design of switching circuits. This major achievement in the area of switching theory should be accredited to C. E. Shannon (3), who recognized for the first time that each of the three fundamental algebraic operators, AND, OR, and NOT, can be represented by actual logic hardware. Shannon demonstrated that Boolean algebra provides a mathematical means for simplifying a switching circuit.

As the evolution of switching theory progresses, a need for a more specialized algebra is established. This need is reflected by the development of non-conventional logic elements, such as NOR, NAND, INHIBITOR, EXCLUSIVE OR, and COINCIDENCE elements. Although an algebraic representation can be made for each of these elements, it is the author's opinion that a generalized algebra, which includes each available logic element as an algebraic operator would be most advantageous in designing circuits which utilize these special types of elements. This advantage will become more apparent when computers are used for aiding the design of the logic networks.

The following section offers a foundation for the development of such an algebra. Even though the completeness of the algebra might be challenged, its applicability for some non-conventional logic devices demonstrates its practical value in switching circuit theory.

4.2 The Algebra

Switching circuit algebra is a mathematical system consisting of variables, operators, constants, and an equivalence; which are governed by a set of postulates defining the algebra. The algebra concerns itself with variables having only two values, which are the constants of the algebra, $\emptyset$ and U. It employs a set of symbols which are the variables of the algebra; upon which manipulations are performed by four commutative and distributive (or conventional) operators, two non-conventional operators, and one complementing function. The term "non-conventional" operators is used to denote operators which are neither commutative nor distributive. For example, the operators AND (.) and OR (+) are conventional operators; while INHIBITOR's and EXCLUSIVE OR's can be classified as non-conventional operators. Let the four conventional operators be represented by the four arbitary symbols, o, ¢, @, and Ø; and let the non-conventional operators be represented by the symbols ν and η. As also commonly used in Boolean algebra, the symbols =, ≠, and ' are used to represent equivalence, non-equivalence, and complementation, respectively.

The algebra is defined by the following postulates:

<u>Postulates</u>

	<u>Equation</u>		<u>Dual</u>
1.	oX = oU if oX ≠ oØ	1.	¢X = ¢Ø if ¢X ≠ ¢U
2.	@X = @Ø if @X ≠ @U	2.	ØX = ØU if ØX ≠ ØØ
3.	o(@U,@U) = o@U	3.	¢(ØØ,ØØ) = ¢ØØ
4.	@(oØ,oØ) = @oØ	4.	Ø(¢U,¢U) = Ø¢U
5.	o(@U,@Ø) = o@Ø	5.	¢(ØU,ØØ) = ¢ØU
6.	o(U') = o(Ø)	6.	¢(Ø') = ¢(U)
7.	@(Ø') = @(U)	7.	Ø(U') = @(Ø)
8.	η(U,U) = Ø	8.	ν(Ø,Ø) = U
9.	η(Ø,Ø) = Ø	9.	ν(U,U) = U
10.	η(U,Ø) = U	10.	ν(Ø,U) = Ø

In order to aid the user of the algebra with mathematical manipulations, theorems are developed. Unlike the postulates, theorems must be derived entirely from the postulates; they cannot contain any assumptions which are not reflected by the postulates. The following theorems are considered important in switching theory:

Theorems

1.	Tautology	a.	o(X,X) = o(X)
		b.	¢(X,X) = ¢(X)
		c.	@(X,X) = @(X)
		d.	@̸(X,X) = @̸(X)
2.	Commutative	a.	o(X,Y) = o(Y,X)
		b.	¢(X,Y) = ¢(Y,X)
		c.	@(X,Y) = @(Y,X)
		d.	@̸(X,Y) = @̸(Y,X)
3.	Association	a.	o(@̸X,@̸(Y,Z)) = o(@̸(X,Y),@̸Z)
		b.	¢(@X,@(Y,Z)) = ¢(@(X,Y),@Z)
		c.	@(¢X,¢(Y,Z)) = @(¢(X,Y),¢Z)
		d.	@̸(oX,o(Y,Z)) = @̸(o(X,Y),oZ)
4.	Distribution	a.	o(@X,@(Y,Z)) = ¢(@̸(X,Y),@̸(X,Z))
		b.	¢(@̸X,@̸(Y,Z)) = o(@(X,Y),@(X,Z))
		c.	@(oX,o(Y,Z)) = @̸(¢(X,Y),¢(X,Z))
		d.	@̸(¢X,¢(Y,Z)) = @(o(X,Y),o(X,Z))
5.	Absorption	a.	o(@X,@(X,Y)) = o@(X)
		b.	¢(@̸X,@̸(X,Y)) = ¢@(X)
		c.	@(oX,o(X,Y)) = @o(X)
		d.	@(¢X,¢(X,Y)) = ¢%(X)

6.	Inclusion	a.	o(¢X,¢(X,Y)) = o¢(X)
		b.	¢(oX,o(X,Y)) = ¢o(X)
		c.	@(@̸X,@̸(X,Y)) = @@̸(X)
		d.	@̸(@X,@(X,Y)) = @̸@(X)
7.	Universe Class	a.	o(@X,@U) = o@(X)
		b.	¢(@̸X,@̸U) = ¢@̸(U)
8.	Null Class	a.	o(@X,@Ø) = o@(Ø)
		b.	¢(@̸X,@̸Ø) = ¢@̸(X)
9.	Complementation	a.	o(@X,@X') = o@(Ø)
		b.	¢(@̸X,@̸X') = ¢@̸(U)
10.	Contraposition	a.	If o(X) = o(Y'), then o(X') = o(Y)
		b.	If ¢(X) = ¢(Y'), then ¢(X') = ¢(Y)
		c.	If @(X) = @(Y'), then @(X') = @(Y)
		d.	If @̸(X) = @̸(Y'), then @̸(X') = @̸(Y)
11.	Double Negation	a.	o(X") = o(X)
		b.	¢(X") = ¢(X)
		c.	@(X") = @(X)
		d.	@̸(X") = @̸(X)
12.	Expansion	a.	o(@(X,Y),@(X,Y')) = o@(X)
		b.	¢(@̸(X,Y),@̸)X,Y')) = ¢@̸(X)
		c.	@(o(X,Y),o)X,Y')) = @o(X)
		d.	@̸(¢(X,Y),¢(X,Y')) = @̸¢(X)
13.	DeMorgan's Theorem	a.	o(X,Y)' = ¢(X',Y')
		b.	¢(X,Y)' = o(X',Y')
		c.	@(X,Y)' = @̸(X',Y')
		d.	@̸(X,Y)' = @(X',Y')
		e.	η(X,Y)' = ν(X',Y')

		f.	$\nu(X,Y)' = \eta(X',Y')$
14.	Reflection	a.	$o(@X,@(X',Y)) = o(@X,@Y)$
		b.	$\text{¢}(\cancel{@}X,\cancel{@}(X',Y)) = \text{¢}(\cancel{@}X,\cancel{@}Y)$
		c.	$@(oX,o(X',Y)) = @(oX,oY)$
		d.	$\cancel{@}(\text{¢}X,\text{¢}(X',Y)) = \cancel{@}(\text{¢}X,\text{¢}Y)$
15.	Transition	a.	$o(@(X,Y),@(X',Z),@(Y,Z)) = o(@(X,Y),@(X',Z))$
		b.	$\text{¢}(\cancel{@}(X,Y),\cancel{@}(X',Z),\cancel{@}(Y,Z)) = \text{¢}(\cancel{@}(X,Y),\cancel{@}(X',Z))$
		c.	$@(o(X,Y),o(X',Z),o(Y,Z)) = @(o(X,Y),o(X',Z))$
		d.	$\cancel{@}(\text{¢}(X,Y),\text{¢}(X',Z),\text{¢}(Y,Z)) = \cancel{@}(\text{¢}(X,Y),\text{¢}(X',Z))$
16.	Equivalence	a.	$\eta(X,X) = \emptyset$
		b.	$\nu(X,X) = U$
17.	Transposition	a.	$o(@(X,Y),@(X',Z)) = @(o(X,Z), o(X',Y))$
		b.	$\text{¢}(\cancel{@}(X,Y),\cancel{@}(X',Z)) = @(\text{¢}(X,Z), \text{¢}(X',Y))$
		c.	$@(o(X,Y),o(X',Z)) = o(@(X,Z), @(X',Y))$
		d.	$\cancel{@}(\text{¢}(X,Y),\text{¢}(X',Z)) = \text{¢}(\cancel{@}(X,Z), \cancel{@}(X',Y))$

Up to this point, the algebra has been presented in a generalized manner. No attempts were made to assign specific operators or constants to replace the six operators and the two values of the algebra. It is maintained by the author, that there exist many sets of numerals and operators which satisfies the algebra. Among them, a few known combinations of logical operators and constants are listed in Table II and Table III. Note that the entries of each row of these tables must be used in its entirety. For example, when "o" is replaced by the "OR" symbol, "U" must represent the logical constant "O" (see Row 2 of Table II). It should also be noted, that in these tables the notations $\downarrow$, $\uparrow$, INH, IMP, $\oplus$, $\equiv$, are used to represent the logical operators NOR, NAND, INHIBITOR, IMPLY, EXCLUSIVE-OR, and COINCIDENCE, respectively.

4.3 Effects of the New Algebra in Switching Circuit Theory

The application of the new algebra in switching circuit theory provides a new insight into formal combinational logic synthesis. In particular, the feature of having arbitrary operators permits the designer to perform algebraic manipulations without concern of the types of the operators; in other words, the manipulations of the equations can be performed independently from the actual operators that are utilized. This feature is especially practical for the computer-aided design of logic systems.

For example, the Karnaugh Map simplification method can be generalized to satisfy the theorems and postulates of the algebra. For this purpose, the Karnaugh map can be defined as a map which consists of cubes, each of which is

TABLE II

CONVENTIONAL OPERATORS OF THE ALGEBRA

SET No.	o	¢	@	$\not@$	U	O
1	.	+	+	.	1	0
2	+	.	.	+	0	1
3	↓	↑	↓	↑	1	0
4	↑	↓	↑	↓	0	1

TABLE III

NON-CONVENTIONAL OPERATORS

SET No.	η	ν	U	O
1	INH	IMP	1	0
2	IMP	INH	0	1
3	⊕	≡	1	0
4	≡	⊕	0	1

represented by a unique combination of the variables of the logic system. A logic operator governs every variable representation of each cube, while another operator defines the relationship between one cube and another. A set consisting of all available cubes (or the universe of the map) represents a constant of the algebra, while the other constant is defined by the empty set. The valid combinations of operators and constants which can be utilized in the generalized Karnaugh map are listed in Table IV. Similar to the previous identical tables, each row of Table IV reflects a valid combination of operators and constants and hence, it must be used in its entirety.

TABLE IV

KARNAUGH MAP OPERATORS AND CONSTANTS

Set No.	Lines	Cube	Universe	Empty
1	o	@	∅	U
2	¢	$\cancel{@}$	U	∅

As an illustration, the generalized Karnaugh Map is utilized to simplify the following expression:

$$Z = o(@(A,B,C),@(A',B,D),@(A,B,C')) \qquad (4\text{-}1)$$

Without considering the actual logical operators, Equation (4-1) can be projected on the Generalized Karnaugh Map, resulting in the map shown in Figure 1. As the "Cube" representation in the above equation is an "@", while the "Line" representation (the representation between cubes) is an "o", it is established that the universe of the map is "∅" while the empty map correspond with an "U". The simplified solution can be directly derived from the map; which is:

$$Z = o(@(A,B),@(B,D)) \qquad (4\text{-}2)$$

When both "@" and "o" are designated NOR's ($\downarrow$), the universe of the map is "0", while "1" is reflected by the empty map. Interpretation of Equation (4-2) is simply:

$$Z = \downarrow(\downarrow(A,B),\ \downarrow(B,D)) \qquad (4\text{-}3)$$

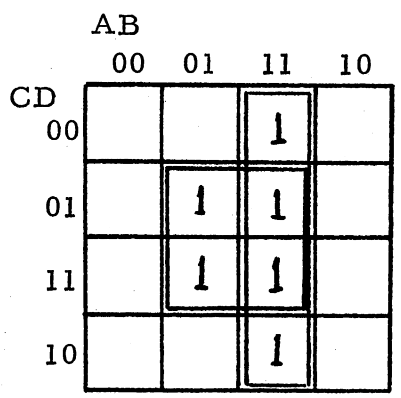


Figure 1. Karnaugh Map of Equation (4-1)

A similar approach can be made when dealing with non-conventional operators. For example, the following equation:

$$\eta(X,Y)' = \nu(X',Y') \qquad (4\text{-}4)$$

(deMorgan's Theorem)

can be interpreted as either:

$$\mathrm{INH}(X,Y)' = \mathrm{IMP}(X',Y') \qquad (4\text{-}5)$$

or:

$$\oplus(X,Y)' = \equiv(X',Y') \qquad (4\text{-}6)$$

The above demonstrations show the flexibility of the algebra. It eliminates the necessity of individual synthesis approaches for each set of operators. Although no meaningful synthesis approach has been explored for the effective utilization of the non-conventional operators, a quite promising direction for future investigations has been established.

CHAPTER V

COMBINATIONAL LOGIC SYNTHESIS

5.1 Philosophy of the Synthesis

Logic synthesis is the process of constructing a desired network based upon a given set of instructions. When the resulting network does not require the utilization of memories, the synthesis is termed as combinational.

In general, there are three objectives which are to be achieved in a logic synthesis. These are:

1. Minimal element networks
2. High speed networks
3. Correct and dependable operation.

The importance of these three goals is apparent. Minimal element networks offers many advantageous features, such as low hardware and operating costs, the possibility of obtaining higher speed networks and convenience in analyzing the network. It should be realized, however, that in most cases increasing the speed of a network depends upon its implementation.

In relation to this study, the combinational synthesis process includes the selection of the appropriate techniques in order to apprach the above objectives. Various useful techniques and their particular applications are discussed in this chapter.

5.2 The Consensus or the *-Product

The consensus approach was introduced for the first time by W. V. Quine (31) in 1952. His approach was extended in a unique format by J. P. Roth (32, 33) in 1955. The method utilizes a combination of the Reflection, the Expansion, and the Transition theorems. The product is represented as follows:

$$f_c = f_a * f_b \qquad (5\text{-}1)$$

where f_a, f_b and f_c are algebraic terms, each of which is represented by n literals as the following:

$$f_a = o(a_1, a_2, \ldots, a_n) \qquad (5\text{-}2a)$$

$$f_b = o(b_1, b_2, \ldots, b_n) \qquad (5\text{-}2b)$$

$$f_c = o(c_1, c_2, \ldots, c_n) \qquad (5\text{-}2c)$$

where "o" is a conventional operator. Each variable, a_i, b_i and c_i can be represented by either a "0", a "1", or an indeterminate variable value, which is indicated by a "-". The product is performed one variable at a time, the results of which is best represented by the table shown in Table V. The rows of this table are represented by the values of a_i, while the columns are represented by the values of b_i.

TABLE V

TABLE FOR $c_i = a_i * b_i$

c_i		b_i 0	1	–
a_i	0	0	φ	0
	1	φ	1	1
	–	0	1	–

The entries in the table are the values of the product, c_i, which are four-valued, e.g., 1, 0, –, and φ. After the product has been applied to each variable of the expression, the composite of the results can be interpreted by the following rules:

1. $a * b = \Phi$ or empty, if $a_i * b_i = \varphi$ for more than one i.
2. $a * b = c$ if otherwise, where c is represented by $(c_1, c_2, \ldots, c_n)$, replacing the "φ" by a "–".

As an illustration, consider for example the *-product between the following two algebraic terms:

$$Z = o(A,B',C',D,F) * o(B,C',E,F,G,H) \qquad (5\text{-}3)$$

which can be represented in the numerical form as follows:

$$Z = o(1001\text{-}1\text{--}) * o(\text{-}10\text{-}1111) \qquad (5\text{-}4)$$

One at a time assessment of each variable results in the following:

$$Z = o(1\varphi011111) \tag{5-5}$$

As this expression has only one "φ", using Rule 2, Z can be interpreted as:

$$\begin{aligned} Z &= o(1-011111) \\ &= o(A,C',D,E,F,G,H) \end{aligned} \tag{5-6}$$

It can be noted that the variable-by-variable assessment as performed in this synthesis approach offers unlimited possibilities of the method in solving large, multi-variable systems. Roth realized that an iterative application of the operation under certain conditions would result in the development of all prime implicants; from which near minimal forms can be obtained. The approach soon became popular as it is the only simplification approach that can tolerate switching functions with extremely large numbers of variables. This is evident from the numerous attempts towards the perfection and the computerization of the approach (34, 35, 36, 37, 38). One such method which was developed by the author has been successfully computerized and is capable of deriving both the minimal and the minimal static hazard free solutions of large algebraic expressions (38).

5.3 Complementation and Distribution

Complementation and distribution of switching circuit equations has been shown to be effective in the simplification of logic networks in many ways (39). They provide means not only for exploring equivalent expressions in different forms, but also for the identification of unknown machine states. It is therefore realized, that a computer-oriented method capable of performing such operations is very valuable.

The Complementation and Distribution operations can be best represented by the Karnaugh map as shown in Figure 2.

	AB			
CD	00	01	11	10
00	1	1	1	1
01	1	0	0	1
11	1	0	0	0
10	1	1	0	0

Figure 2. Karnaugh Map of Equation (5-7)

The generalized expression represented by the map is:

Z = o(@(C',D'),@(A'B'),@(A,B',C'),@(A',C,D')) (5-7)

As it is apparent from the map, the complement of the expression is:

$$Z' = o(@(B,D),@(A,C)) \qquad (5\text{-}8)$$

Performing the DeMorgan's operation upon Z' will result in a solution that is identical to the Distributive equivalence of Z, which is:

$$Z = \not{c}(\not{@}(B',D'),\not{@}(A',C')) \qquad (5\text{-}9)$$

While the Karnaugh Map is an effective aid in generalizing both the complement and the distributive equivalent solution of small switching equations, it loses its potential practicality when dealing with large, multivariable equations. Fortunately, mechanized methods exist; and one method which is capable of generating the prime implicants of the complement and those of the distributive equivalent has been perfected and successfully computerized (40).

5.4 The Term Simplification or the &-Product

The term simplification operation or the "&-Product" is an operation which involves one term and a group of terms, and which performs the maximal simplification of the first term as such that it does not conflict with the terms contained in the group. The operation was initially conceived by J. B. Surjaatmadja and E. C. Fitch (41) in 1975. The product can be represented as:

$$A = a \ \& \ B \tag{5-10}$$

where "A" and "a" are terms of n variables and "B" is a set of terms, all of the same type; e.g., only one operator is used for representing each of the terms. When projected in the Karnaugh Map, the "&-Product performs a critical expansion of the term "a" cube as such that it will not intersect the "B" cube (see Figure 3).

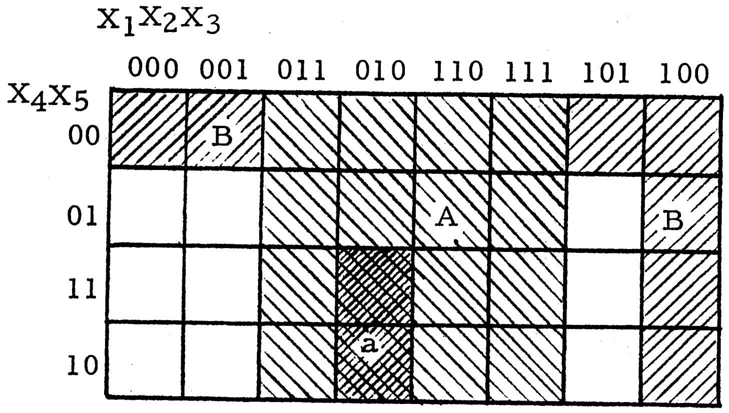


Figure 3. Karnaugh Map Showing the &-Product

The product is best performed using a "clause table" format as follows:

$$(A_1, A_2, \ldots, A_n) = (a_1, a_2, \ldots, a_n) \mathbin{\&} \begin{bmatrix} B_{11}, B_{12}, & \ldots, & B_{1n} \\ B_{21}, B_{22}, & \ldots, & B_{2n} \\ \vdots & & \vdots \\ B_{m1}, B_{m2}, & \ldots, & B_{mn} \end{bmatrix} \quad (5\text{-}11)$$

A clause table is an array representation of the terms without the inclusion of their operators. Each term is represented by a row, while each column relates to a variable of the system. Unrepresented variables are listed as "-"s, denoting indeterminate variable values. The operation is most conveniently performed in two steps:

1. Modify the "B" matrix by replacing each Bij-element (or literal) by element a_j, provided by B_{ij} is not equal to a_j and none of the two elements is a "-".
2. Select a minimal combination of variables (columns) such that each term (or row) of "B" is represented at least once by the combination. This minimal combination represents the result of the &-product.
3. The product is termed as unsuccessful if no such combination exists; in other words, the product will not be successful if one of the rows of the modified B-matrix is empty.

As an illustration of the procedure, consider the following equations:

$$a = @(X_1, X_2', X_3, X_4', X_5, X_6') \quad (5\text{-}12)$$

$$B = o(@(X_1', X_2', X_3'), @(X_3, X_4, X_5), @(X_2, X_5, X_6)) \quad (5\text{-}13)$$

The matrix representation of B would be:

$$B = \begin{pmatrix} X_1' & X_2' & X_3' & - & - & - \\ - & - & X_3 & X_4 & X_5 & \\ - & X_2 & - & - & X_5 & X_6 \end{pmatrix} \quad (5\text{-}14)$$

Modification of the "B" matrix relative to "a" gives:

$$B = \begin{pmatrix} X_1 & - & X_3 & - & - & - \\ - & - & - & X_4' & - & - \\ - & X_2' & - & - & - & X_6' \end{pmatrix} \quad (5\text{-}15)$$

It can be verified that a minimal coverage of all B-terms is obtained by three variables, which can be satisfied by four solutions:

$$A = @(X_1, X_4', X_6') \quad (5\text{-}16a)$$

$$= @(X_1, X_2', X_4') \quad (5\text{-}16b)$$

$$= @(X_2', X_3, X_4') \quad (5\text{-}16c)$$

$$= @(X_3, X_4', X_6') \quad (5\text{-}16d)$$

Each of these solutions is the minimal reduction of term "a" with respect to "B".

5.5 The Synthesis of Single Terminal Networks

There are unlimited possibilities as to how a designer can formulate a desired network. However, it is generally accepted that simplification of the network expression would at least provide an intermediate step towards the optimal reduction of the network implementation. This is especially true for networks having only one output, where optimal simplification of the output expression would generally lead towards a near minimal network implementation.

For the simplification of functions having less than 6 variables, the use of Karnaugh Maps for performing the synthesis is irrefutable. However, for systems with higher numbers of inputs the utilization of maps is not practical. For such systems, the application of the consensus approach together with the complementation approach has been proven to be very successful. In this approach, the consensus is used to generate the prime implicants of the network equations while complementation produces the don't cares of the system. The practicality of the method can be demonstrated using the 10-variable example problem presented in the Truth Table of Table VI.

TABLE VI

TRUTH TABLE

a	b	c	d	e	f	g	h	i	j	Z
1	0	1	0	1	0	1	0	1	0	1
1	1	0	1	1	0	1	0	1	1	1
0	1	0	1	1	0	1	-	1	1	1
1	0	0	1	0	1	0	0	1	0	0
0	1	0	0	1	0	1	0	1	0	0

It is assumed that all other unknown machine states are don't cares. The representation of Z is as follows:

$$Z = o(@(a,b',c,d',e,f',g,h',i,j'),@(a,b,c',d,e,f',g,h',i,j),@(a',b,c',d,e,f',g,i,j)) \quad (5\text{-}17)$$

It can be verified that the only consensus operation that can be performed is between the second and the third term, which results in:

$$Z_2 = @(b,c',d,e,f',g,h',i,j) \quad (5\text{-}18)$$

which replaces the second term of Equation (5-17).

Furthermore, complementation of the Z' function gives the expression of both the don't cares and the Z-expression in combination. The complement can be derived to and will result in the following:

$$Z_{d.c.} = o(@(c,e),@(d,e),@(a,b),@(a',b'), \ldots. \quad (5\text{-}19)$$

By selecting only the necessary terms of $Z_{d.c.}$ needed to cover Z, the minimal representation of Z is as follows:

$$Z = o(@(c,e),@(d,e)) \quad (5\text{-}20)$$

which shows a substantial reduction of Z. The absolute minimal implementation is obtained by applying the Distribution theorem; resulting in the following minimal expression:

$$Z = ¢(\not@e,\not@(c,d)) \quad (5\text{-}21)$$

If Z is an AND-OR-NOT relationship, with an AND for ¢ and an

OR for $\not{p}$, the circuit representation is the two-element circuit shown in Figure 4.

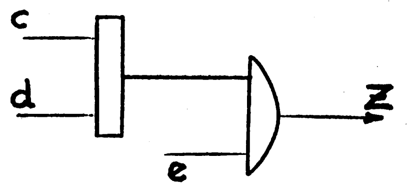


Figure 4. Circuit Representation of the System Represented in Table VI

5.6 The Synthesis of Multi-Terminal Networks

When the network that is to be synthesized has more than one output, individual minimization of each output may fail to produce a minimal network (39). Under such circumstances, the importance of synthesizing the network as one unified system cannot be ignored.

There are various techniques available for performing multi-terminal network synthesis (43, 43, 44, 45). However, the application of these methods in Fluid Logic pose a serious problem; as the success of these methods depends upon the utilization of unlimited fan-in elements--a feature not generally found in fluid logic elements. Therefore, it is important that the selected simplification method be constructed as such that it is effective for limited fan-in elements as well.

The method advanced in this research effort performs the simplification using the following steps:

1. For each machine state, group the outputs which have an ON or "1" state.
2. Using the &-Product, simplify each input state with respect to all "Zero states" of the associated group of outputs. Classify as "Zero States" all input states which have a zero output for one or more outputs of the group under consideration.
3. Tabulate the coverage of each simplified term over the original terms. Here, Term "A" is said to cover Term "B" if all elements of "A" are contained in "B".
4. Based upon the term coverage, select a minimal combination of terms to represent each output.

As an illustration, consider the "Truth Table" as represented by the first three columns of the Multi-Terminal Synthesis Table of Table VII. The first term of this table produces only one output, e.g., Z_3. Therefore, the "group" of outputs relating to the first input state contain only Z_3; and as the zero states of Z_3 are States 4, 5, 6, and 7, the &-Product of State 1 must be performed relative to these zero states; which results in the term "a" (see Column 4).

TABLE VII

MULTI-TERMINAL SYNTHESIS TABLE

No.	Outputs			Input States						Simplified Term						Term Coverage
	Z_1	Z_2	Z_3	a	b	c	d	e	f	a	b	c	d	e	f	Term No.
1	0	0	1	1	0	1	0	1	0	1	-	-	-	-	-	1
2	0	1	1	0	0	1	0	1	0	0	0	-	-	-	-	2,8
3	1	1	1	0	1	1	0	1	0	-	1	1	-	-	-	3
4	1	1	0	0	1	0	0	1	0	-	-	0	0	-	-	4,8
5	1	0	0	0	1	0	1	1	0	-	-	-	1	1	-	5
6	0	0	0	0	1	0	1	0	0	-	-	-	1	0	0	6
7	1	1	0	0	1	0	1	0	1	-	-	-	-	-	1	7
8	1	1	1	0	0	0	0	0	0	-	0	0	-	-	-	8

Similarly, State (2) has two outputs -- Z_2 and Z_3 -- and hence the &-Product is performed against States 1, 4, 5, 6, and 7. The simplified terms are tabulated in the fourth column of Table VII.

After the fourth column has been completed, the next important step is the accounting of the term coverage of each simplified term. As an illustration consider the second simplified term of Table VII. Because of the fact, that ab = 00 is contained in Terms 2 and 8 (see Column 2), this simplified term is said to cover both Terms 2 and 8; and this information is tabulated in Column 5.

The final task is the selection of the minimal representation of the network. This selection can be performed by using any of the available accounting techniques; which results in the following equations:

$$Z_1 = o(@(b,c),@(c',d'),@(d,e),@(f)) \qquad (5\text{-}22a)$$

$$Z_2 = o(@(a',b'),@(b,c),@(c',d'),@(f)) \qquad (5\text{-}22b)$$

$$Z_3 = o(@(a),@(a',b'),@(b,c))$$

$$= o(@(a),@(b'),@(c)) \qquad (5\text{-}22c)$$

As the third term of Equation (5-22c) also covers the first machine state, the Term "a" is redundant, and hence it can be eliminated. Furthermore, in order to avoid individual complementation of "b", the earlier form of the second term should be retained; which gives the following simplified version of Equation (5-22c):

$$Z_3 = o(@(a',b'),@(c)) \qquad (5\text{-}22d)$$

When the expressions of Z_1, Z_2, Z_3 are conventional disjunctive equations, only nine elements are required for their implementation. The network representation is shown in Figure 5. Note that the application of individual synthesis may result in a more simplified expression; however, the implementation of such expressions do not always result in a simplified network. For example, by individual synthesis, the above example will produce:

$$Z_1 = o(@(b',e'),@(b,e),@(f)) \qquad (5\text{-}23a)$$

$$Z_2 = o(@(b',e'),@(a',c),@(c',d'),@(f)) \qquad (5\text{-}23b)$$

$$Z_3 = o(@(c),@(b')) \qquad (5\text{-}23c)$$

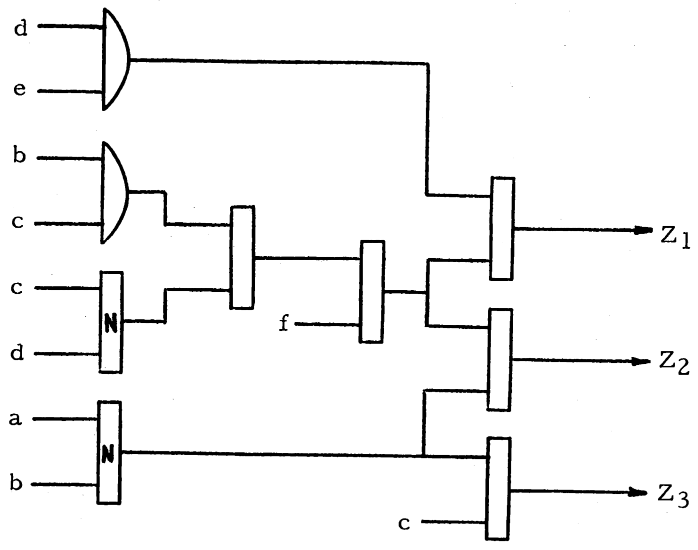


Figure 5. Network Representation of Equation (5-22)

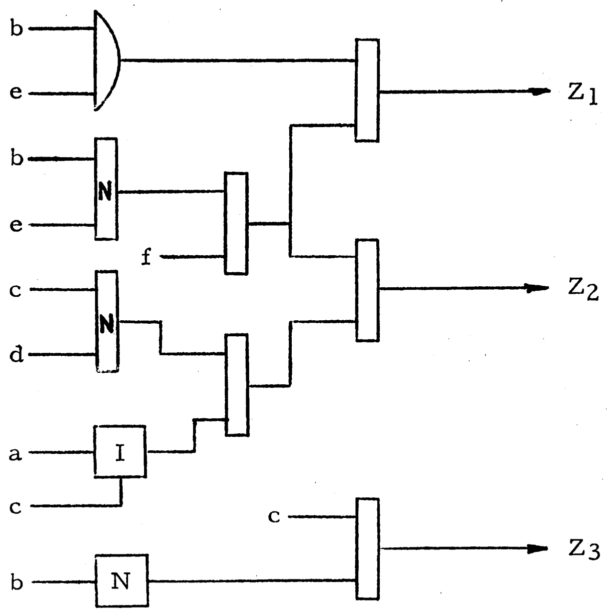


Figure 6. Network Representation of Equation (5-23)

Implementation of the above equations shows that the simplicity of the equations are not reflected in the actual network construction as shown in Figure 6.

5.7 Three-Level Synthesis of NOR-Logic Networks

NOR logic elements play an important role in Fluid Logic. This is attributed to the fact, that NOR elements are not only functionally complete, but also that they are basically constructed to have many inputs--a feature which is a key towards both simplification and increasing circuit speed.

The synthesis of NOR logic networks requires special attention. It is believed, that a minimal network configuration can be obtained when the circuit is constructed with three levels--the complementation level, the conjunction level, and the disjunction level. The term "level" is used to represent the number of elements through which an input signal to an element under consideration must transgress in order to reach the output.

Basically, a three-level network can be obtained from a two-stage expression as follows:

$$Z = o(@(X_1, X_2, \ldots), @(X_n, X_m, \ldots) \ldots) \qquad (5\text{-}24)$$

where each X_i can be substituted by a complemented, uncomplemented, or indeterminate variable value; and where both operators o and @ are NOR operators. While the implementation of such expressions is quite obvious, it is a fact that the commonly found problem descriptions are not given in NOR forms. In fact, most problem descriptions are given in disjunctive, AND-OR-NOT forms. In such cases, the NOR-forms can be developed using the following steps:

1. Simplify the "0" states of the disjunctive function; using all unspecified machine states as don't cares.
2. Using the DeMorgan's law, complement the expression in order to obtain its complementary conjunctive expression.
3. The NOR-form can be constructed directly from the resulting conjunctive equation, by replacing all operators by "$\downarrow$"s.

In order to illustrate this procedure, consider the truth table shown in Table VI. Having assumed that the terms represented in the table are conventional product terms, the equation of the complement Z' is:

$$Z' = +(\cdot(a,b',c',d,e',f,g',h',i,j'),\cdot(a',b,c',d',e,f',g,h',i,j')) \qquad (5\text{-}25)$$

The unknown don't cares can be generated by complementing the Z expression, which results in the combination of both the Z' and the don't care expressions as follows:

$$Z'_{d.c.} = +(\cdot(e'),\cdot(f),\cdot(g'),\cdot(i'),\cdot(c',j') \ldots) \qquad (5\text{-}26)$$

Using available accounting techniques, it can be derived that the minimal coverage can be obtained by the fifth term of $Z'_{d.c.}$; and hence:

$$Z' = +.(c',j') \tag{5-27}$$

Complementation using the DeMorgan's law results in:

$$Z = .+(c,j) \tag{5-28}$$

The final step is the substitution of "$\downarrow$" operators for all the operators of Equation (5-28), which becomes:

$$Z = \downarrow\downarrow(c,j) \tag{5-29}$$

which can be represented by the two element network shown in Figure 7.

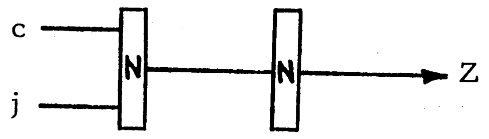


Figure 7. Network Representation of Equation (5-29)

Although, in general, near minimal networks are obtained by this approach, in some special types of problems further simplification can be obtained by rearranging the network implementation. A most successful technique has been presented in References (39) and (46), which will not be discussed in this presentation.

CHAPTER VI

SEQUENTIAL LOGIC SYNTHESIS

6.1 Philosophy of the Synthesis

Sequential Logic Synthesis is the process of transforming a sequential problem into one or more combinational logic problems in a most efficient manner. A successful synthesis is one which leads to the development of minimal hardware networks.

As initially conceived by Huffman (6) in 1954, a sequential circuit synthesis is best performed by reducing it to the problem of deriving the intended outputs, called the primary outputs, and the problem of deriving the secondary outputs which are necessary for the activation and deactivation of memory elements needed by the sequential system. Memory elements are required in sequential networks as they are the only means for recording the history of the past machine states--a characteristic feature which distinguishes sequential from combinational networks.

In order to provide a convenient basis for the discussion of the approach, the various steps involved in the synthesis are discussed individually. These are:

1. The formulation of the logic description
2. The selection of the peripheral circuits

3. The simplification of the logic description
4. The assignment of memories
5. The formulation of the network equations.

6.2 The Formulation of the Logic Description

There are various ways a designer can represent his problem: by a Timing Chart, a Primitive Flow Table, a Synthesis Table, a Sequence Matrix, and by a Logic Specification Chart. The utilization of timing charts are restricted to regularly activated or deterministic circuits; while primitive flow tables are not practical for systems with large numbers of variables. This leaves three equally important discriptive methods which will be discussed in this presentation.

The Synthesis Table is the easiest one to construct. It essentially lists all machine states in one column while recording the next possible states in another column. A typical synthesis table is shown in Table VIII. The first column of the table contains the identification numbers of each machine state, while in the next column, the output states are listed. The third column contains the input states of each machine state. Finally, all next possible states of each machine state are listed in Column 4. The remaining section of this table is reserved for performing the synthesis.

TABLE VIII

THE SYNTHESIS TABLE

No.	Outputs			Inputs			Next States	Synthesis
	Z_1	Z_2	Z_3	a	b	c		
1	0	0	0	0	0	0	2, 8	
2	0	0	1	1	0	0	1, 3	
3	0	0	1	1	1	0	4, 6	
4	1	0	1	1	0	0	3, 5	
5	1	1	1	1	0	1	2, 6	
6	1	1	1	1	1	1	5, 7	
7	1	0	1	1	1	0	6, 8	
8	1	0	0	0	1	0	1	

The interpretation of the synthesis table is quite obvious, as it merely is a compilation of states and their associated next states. For example, State 3 has an output state $Z_1'Z_2'Z_3$ and an input state abc'. States 4 and 6 are listed as the next possible states of State 3 following input signal changes of b and c, respectively. Note that conventional product terms are still utilized, as they provide the best logical interactions between the system and its designer.

The Sequence Matrix Representation resembles closely the Synthesis Table, with one major difference where the recording of the next states is performed in a matrix format. In this matrix representation, next states are recorded as "1"s in the columns of the "Next State Matrix" that correspond to the next states under consideration. To illustrate this representation method, the problem of Table VIII is again reflected in Figure 8, but in a sequence matrix format.

$$\begin{array}{c|ccc|ccc|cccccccc}
 & \multicolumn{3}{c|}{\text{Outputs}} & \multicolumn{3}{c|}{\text{Inputs}} & & & \multicolumn{4}{c}{\text{Next States}} & & \\
\text{No.} & Z_1 & Z_2 & Z_3 & a & b & c & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
2 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
4 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
5 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
6 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
7 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
8 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Figure 8. The Sequence Matrix Representation

Comparing Figure 8 to the Synthesis Table of Table VIII, it can be agreed that the output and input matrices of Figure 8 is identical to the second and third column of the synthesis table. The fourth column is transformed into a "Next State Matrix", which is the third matrix of Figure 8. The interpretation of this matrix can be most conveniently performed by observing the locations of the "1" entries in the matrix. For example, the next states of State 5 are States 2 and 6, which are reflected as "1"s in Row 5, Columns 2 and 6.

Although a substantial expansion of the next state representation is apparent in the matrix format, it is realized that the Sequence Matrix is most appropriate for synthesizing by digital computers, as it requires a minimal amount of computer core.

The third logic description medium is the Logic Specification Chart (LSC). An LSC is a modified primitive flow table where the input states are arranged in a random manner. Unused input states are not represented in the chart, as can be seen in the LSC of Table IX. Note that the example problem of Table VIII is utilized for comparison purposes.

The mechanics of the LSC is identical to that of the Primitive Flow Table. Shown in parentheses are the "stable" states of the system, while the non-paranthetical entries indicate the next states or the "unstable states". A machine state is called stable if it does not change state without a change in the inputs.

TABLE IX

THE LOGIC SPECIFICATION CHART

Outputs			Input States (abc)					
Z_1	Z_2	Z_3	000	100	110	101	111	010
0	0	0	(1)	2				8
0	0	1	1	(2)	3			
0	0	1		4	(3)		6	
1	0	1		(4)	3	5		
1	1	1		2		(5)	6	
1	1	1			7	5	(6)	
1	0	1			(7)		6	8
1	0	0	1					(8)

Because of its format, the LSC provides a convenient means for an in-depth observation of the logic system. Its potential as a descriptive means of recording a logic problem has been demonstrated by the many complex procedures which has been developed based upon its forerunner, the primitive flow table.

6.3 The Selection of Peripheral Equipment

In the past, the selection of peripheral equipment has been totally excluded from the synthesis. No attempt was made to select such equipment to be commensurate to the problem description, and the synthesis was performed based upon the preselected input-output circuits. It is realized, however, that the utilization of different types input-output circuits may cause a complete modification of the network function. Therefore, it is the intention of this section to formulate a selection criterion which may result in the development of minimal hardware networks.

While there are different types of input elements that are available, their utilization in fluid logic circuits are basically identical. For moving part input elements, they may be normally closed or normally open three-way valves; or they may even be constructed using four-way valves (see Figures 9a, b). There are also various non-moving part sensing devices, such as the "proximity sensor" and the "interruptable jet sensor" shown in Figures 9c, d.

In the formulation of the logic specification, additional input variables generally would add to the complexity of the specification. Therefore, input reduction schemes have been attempted in the past in order to avoid unnecessary proliferation of the input variables. A most practical scheme can be observed in Figure 10. The following limitation is imposed in order that two input signals can be combined:

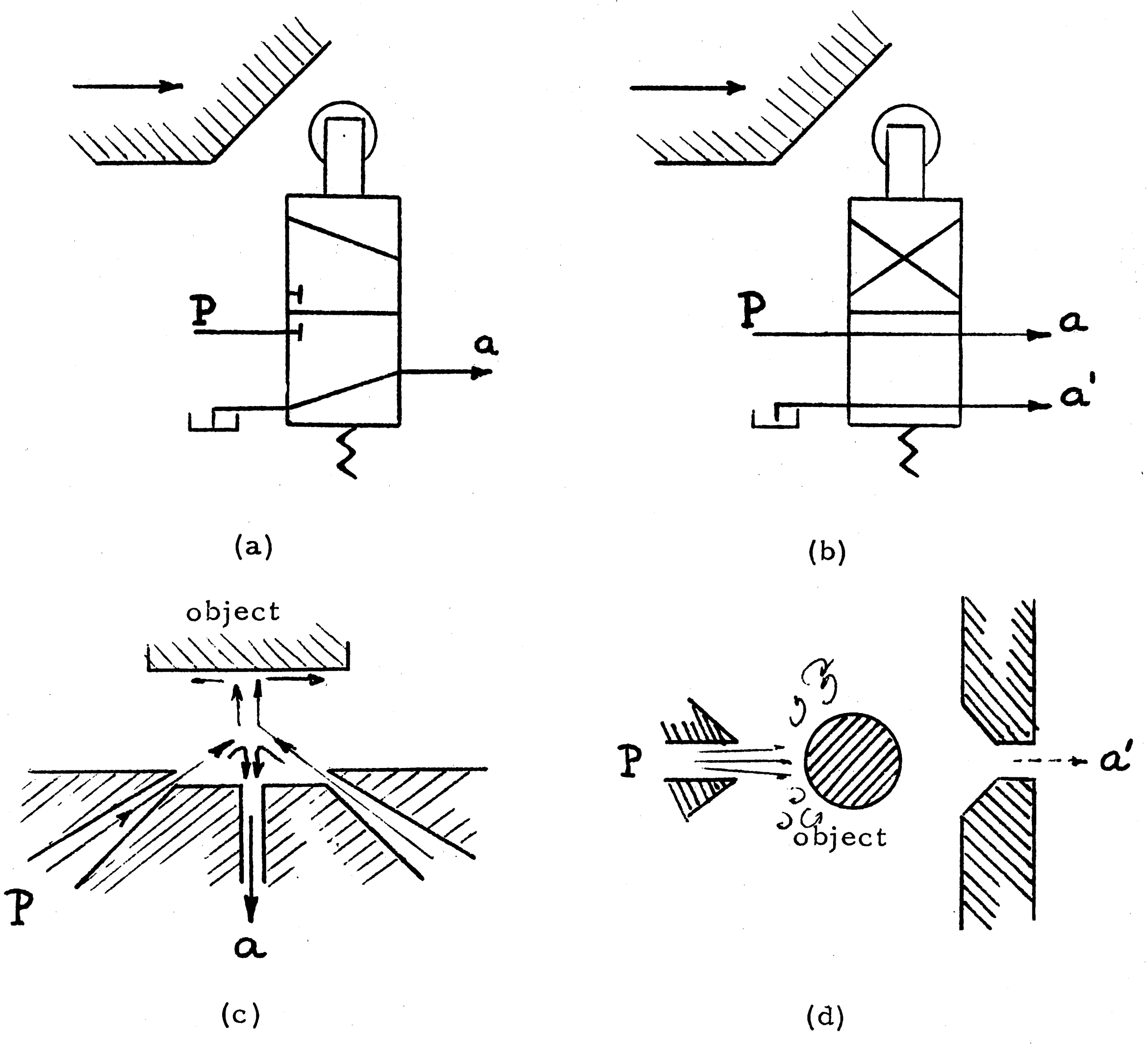


Figure 9. Various Input Sensing Devices

"Two logic signals can be combined to form one single variable and its complement if and only if the two signals never occur simultaneously and if a change of one always leads to the change of the other".

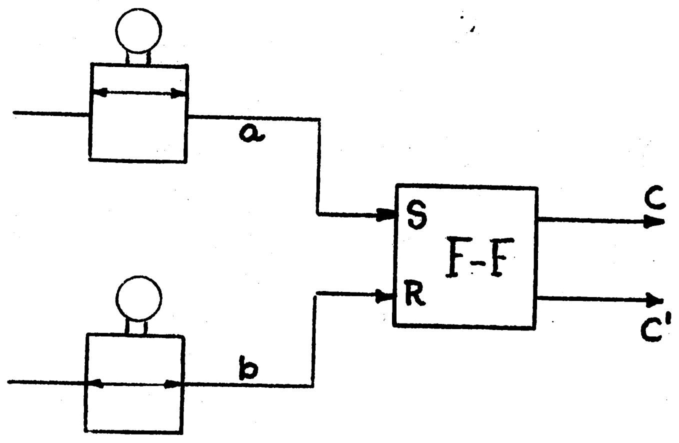


Figure 10. An Input Reduction Scheme

A classic example where such substitution can be performed is the cylinder circuit shown in Figure 11. If the recognized conditions of the cylinder are only its fully extended and fully retracted positions, a new variable can be introduced to replace the two original variables, a and b. It is realized, that the utilization of such input reduction schemes not only reduces the complexity of the logic specification, but also eliminates the need for individual variable complementations of the associated inputs.

While the selection criterion of input circuits are most straightforward, the task of selecting the most suitable type of output circuits is not as simple. This is attributed to the fact that the selection of the output circuits influences the synthesis of the network; and therefore, incorrect selection of such circuits may proliferate the complexity of the implemented network.

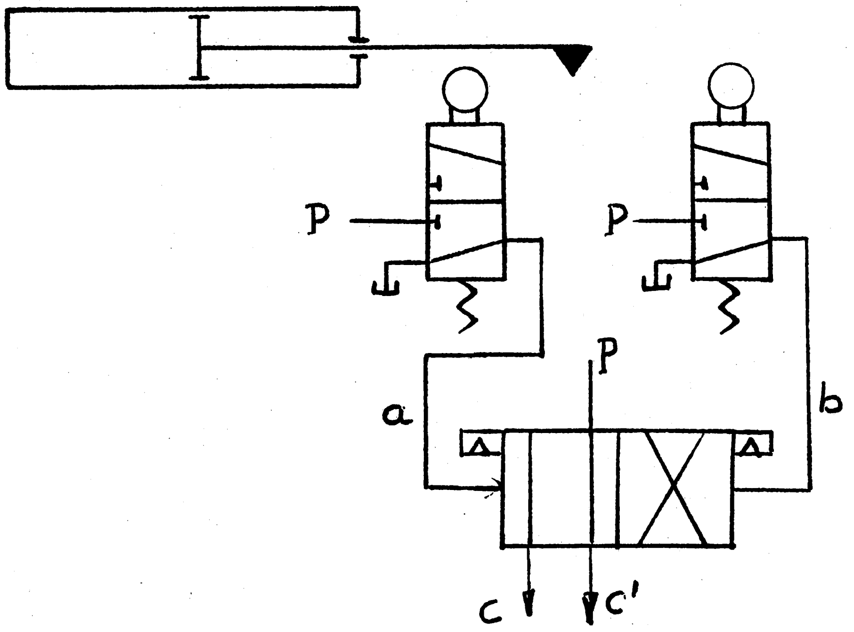


Figure 11. A Typical Input Circuit

Although there are many types of output circuits available, basically two types should be recognized; which are:

a. Output circuits with spring-return actuated power elements,

b. Output circuits with detented power elements.

Basically, the spring-return type power elements transmits power only when it is actuated; while the detented type retains its actuated state until another signal causes it to change. It is obvious that each of these types of circuits has its own effective regions, each type being more advantageous in its designated operating conditions.

In order to enable further discussion concerning the output selection approach, the definition of "active" and "passive" outputs need to be considered. An output is said to be "active" at a particular machine state if the state under discussion may cause the particular output to switch. In other words, an active output is characterized by one or more previous outputs which have a different output state. On the other hand, an output which inherits its current state from its previous states is termed as a passive output.

When the system output exhibits a large number of passive states, the utilization of a detented output element is advantageous as it permits the assessment of the passive states as don't care states. Note, however, that the implementation of such detented output elements requires the generation of two individual signals--the "Set" and "Reset" signals--compared to the single signal configuration when spring-return power elements are used. A trade-off point between the two implementation schemes is, therefore, recognized; which in one argument involves the simplification that is obtained when additional don't cares are available, while the other involves the obvious reduction achieved when a single signal representation is utilized.

Because of the random nature of logic problems, no method exists which actually can predict the degree of simplification that can be gained by each of the above implementation schemes. Fortunately, there are only very few solutions which can be generated by the two schemes; and therefore, direct comparisons between the resulting expressions seem to be quite suitable.

When such a comparative approach is not desired, a statistical means can be conducted for predicting the probable simplification which can be achieved by each implementation scheme. It is known that an association of two terms can either be or not be simplified; and if it is assumed that each case can occur with equal probability, then it is concluded that the utilization of detented output valves would be favorable if the following conditions are satisfied:

$$N_{0a} < N_{1p} \tag{6-1a}$$

$$N_{1a} < N_{0p} \tag{6-1b}$$

where N_{1a} and N_{0a} denotes the number of active "1" and "0" states, respectively; while N_{1p} and N_{0p} are the number of passive "1" and "0" states.

In order to illustrate the latter approach, the two situations as reflected in Table X are considered. In this table, the subscripts "a" and "p" are used to denote the active and passive states of the outputs, respectively.

In Case 1, the number of active "1" and "0" states exceeds the number of passive "0" and "1" states, respectively; and therefore, according to Equations (6-1a, b), the utilization of detented power elements would not be beneficial. This can be demonstrated by the single output representation of Z as follows:

$$Z = +(.(a',c'),.(b',c)) \tag{6-2}$$

while the "Extend"/"Retract" output representation for the detented peripheral element is:

$$Z_E = +(.(a',c'),.(b',c)) \tag{6-3a}$$

$$Z_R = +(.(b,c),.(a,c')) \tag{6-3b}$$

which, when combined, are obviously more complex than the expression of Equation (6-2). When more passive states prevail (such as in Case 2), the utilization of detented output elements may be rewarding. This is evident from the simplification of the "Extend/Retract" expressions of Case 2; which are:

$$Z_E = .(a',c') \tag{6-4a}$$

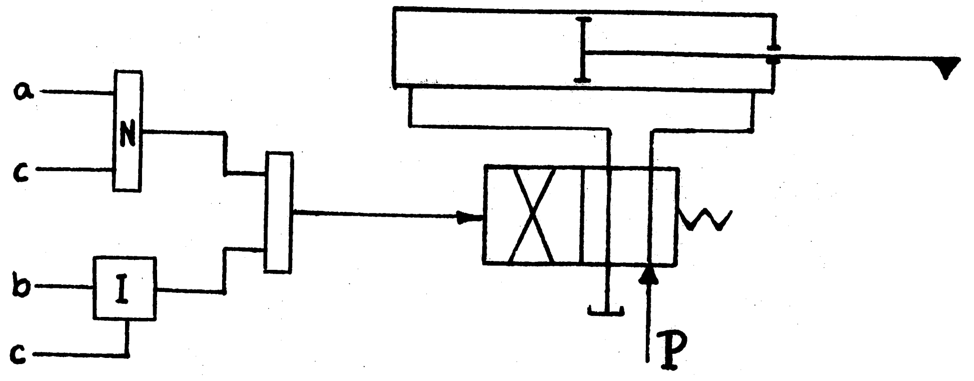
$$Z_R = .(b,c) \tag{6-4b}$$

TABLE X

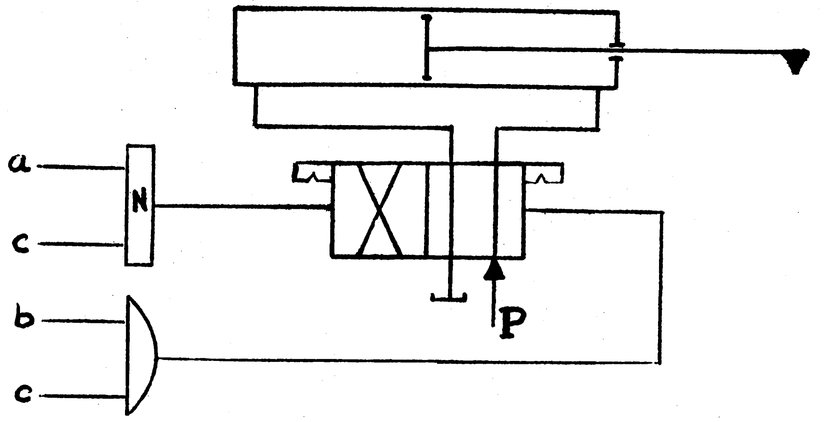
TRUTH TABLE PORTRAYING ACTIVE AND PASSIVE OUTPUTS

Inputs			Case 1	Case 2
a	b	c	Z	Z
0	0	0	1_a	1_a
0	0	1	1_p	1_p
0	1	1	0_a	0_a
0	1	0	1_a	1_a
1	1	0	0_p	0_p
1	1	1	0_p	0_p
1	0	1	1_a	1_p
1	0	0	0_a	0_p

Note that the single output representation remains the same for both cases. Implementation of Equations (6-2) and (6-4a, b) reveals that the utilization of a detented output element for Case 2 results in a more simplified network than when spring-return elements are used (see Figures 12a and b).



(a) Using a Spring-Return Output Element



(b) Using a Detented Output Element

Figure 12. Output Circuit Implementation

6.4 The Simplification of the Problem Description

The simplification or reduction of problem descriptions has been considered as an important step in the synthesis. Such simplification steps may range from a simple elimination of repetitive states, or it may be as extensive as compressing the problem description in order to obtain a minimal memory circuit. The utilization of such reduction schemes has been quite rewarding; not only in reducing the complexity of the problem, but also in reducing the amount of hardware necessary for the final implementation of the network.

The first known reduction method was applied by Huffman (6) in his sequential synthesis technique in 1954. The main purpose of the method is to minimize the number of memory elements necessary for implementing the desired network by minimizing the number of rows in the problem description, which in this case is the primitive flow table. It was contended that such minimal row representation would lead to the development of a minimal element network.

However, such minimal reduction approaches have been contested by many scientists (20, 21, 22, 23, 24, 25, 26, 27, 28, 29) who realized that a minimal row reduction scheme would most likely not result in a minimal element network. Yet, it has been shown that the elimination of unnecessary states is a vital step for reducing the amount of hardware required in the network implementation.

There are two types of machine state elimination schemes. These are:

1. The elimination of all machine states having an "uninfluential input state".
2. The elimination of machine states which can be represented by other states.

The term "uninfluential input states" is used to represent input states which can be either included or excluded from the logic description without altering the logical interpretation of the specification.

When an LSC is used for the logic description of the problem, the first reduction scheme involves the elimination of columns, while the second scheme depicts the deletion of rows of the chart. It is therefore appropriate that the terms "column reduction" and "row reduction" be utilized if such charts are used in the synthesis. However, in order to maintain the generality of the approach, these state reduction schemes are termed as "input state elimination" and "redundant state elimination" in later parts of this presentation.

6.4.1 Input State Elimination

Input state elimination is the process of eliminating a group of machine states which has a common uninfuential input state. Such a process, if successful, obviously would result in a major simplification of the logic description; and therefore an attempt should be made for detecting such possibilities.

The following conditions must be satisfied in order that an input state (or LSC column) can be eliminated:

a. All "circuit outputs" of the machine states which are represented by the particular input state are either don't cares ("-") or zero.

b. There is only one next state listed for each machine state under consideration.

c. For every machine state of the group there should be no previous state that is also the next state.

The term "circuit outputs" is used to denote the outputs which are actually generated by the logic circuit. The term is used to distinguish such types of outputs from the "intended outputs", which may be emitted from the implemented output element.

The input-eliminating procedure can be outlined as follows:

1. Select a group of machine states having an input state (or an LSC column) which satisfies Conditions a, b, and c. If more than one such group exist, select the largest group.
2. Replace every next state entry which is one of the machine states in the group under consideration by its listed next state.
3. Eliminate all machine states represented in the group and rearrange the logic specification.
4. Steps 1, 2, and 3 are performed iteratively until all groups are considered.

It can be agreed that the above conditions and procedures are straightforward. As an illustration, consider the logic specification represented by the LSC shown in Table XI. Assuming that all outputs listed in this table are circuit outputs, it can be observed that all states contained in Column I_2 have zero outputs. Furthermore, it can be verified that every state of Column I_2 satisfies Conditions b and c.

TABLE XI

AN LSC WITH A REDUNDANT INPUT STATE

Outputs		Input States				
Z_1	Z_2	I_1	I_2	I_3	I_4	I_5
1	0	(1)	2		3	
0	0		(2)	4		
0	1	6			(3)	5
1	1		7	(4)		5
0	1	6			3	(5)
1	1	(6)	8		9	
0	0	1	(7)			
0	0		(8)	4		
1	0	1			(9)	10
1	1			4	9	(10)

The next step is modifying the specification in order that the redundancy of Column I_2 is established. This is performed by replacing all next state conditions by their respective next states such that a "lock-up" which excludes all machine states of I_2 is obtained. For example, State 1 has State 2 as its next state. As State 2 is to be eliminated, its substitution by State 4 in Column 3 would provide a "bypassing" condition of State 2. Similarly, State 1 becomes the next state of State 4 as it was the next state of State 7. Elimination of Column I_2 results in the reduced LSC shown in Table XII.

TABLE XII

THE REDUCED LSC

Outputs		Input States			
Z_1	Z_2	I_1	I_3	I_4	I_5
1	0	(1)	4	3	
0	1	6		(3)	5
1	1	1	(4)		5
0	1	6		3	(5)
1	1	(6)	4	9	
1	0	1		(9)	10
1	1		4	9	(10)

It is apparent from the reduced LSC that a major simplification of the chart was obtained. However, the fact that only very few unmodified logic specifications can satisfy the conditions for input state elimination, overshadows the potential effectiveness of the approach. It is, therefore, realized that there exists a critical need for a method capable of recognizing and modifying logic specifications such that it satisfies the input eliminating conditions. Such modifications can be performed by using the following steps:

1. Select an input state which satisfies Input Eliminating Conditions b and c. If more than one such input state exist, select the one representing the largest group of machine states. For identification purposes, let this input state be called "Input State A".
2. Inspect each output variable individually and determine whether or not it satisfies Condition a. If this condition is satisfied, proceed to the next output variable.
3. For each output which does not satisfy Condition a, observe whether or not there are active output conditions represented by Input State A. If such active outputs are present, proceed to Step 6.
4. Replace the existing output element by a detented output element.
5. Consider the next output variable and return to Step 2. The procedure is completed if all outputs have been considered.
6. In the group of states under consideration, observe whether or not all output conditions are "1"s. When only "1" states are found, replace the existing output element by a spring return,

normally open (passing) output element.

7. If both "1" and "0" states are found, the respective input state cannot be eliminated. Proceed to consider the next input state and return to Step 1.

8. Otherwise, consider another output variable and return to Step 2.

Basically, these modifications are concerned with the selection of the proper output circuits in order that the input state elimination can be performed. For example, the utilization of detented elements permits the assessment of all passive states as zero or don't care states, which is a necessary condition in the input elimination process. Step 6 of the above procedure reflects a "primitive" complementation approach for the generation of the intended zero states.

In order to illustrate the procedure, consider the example problem represented by the LSC shown in Table XIII. If it is assumed that the LSC was constructed in relation to the spring return shown in Figure 13, then the value of Z_1 = 1 indicates the extension of cylinder Z_1 while a zero relates to the retraction process.

The active and passive outputs can be determined by careful observation of Table XIII; underlining every output which has a different previous state. Further inspection on this table reveals that Column I_2 is the only column which satisfies Input Eliminating Conditions b and c. However, it is also realized that both outputs do not satisfy Condition

TABLE XIII

THE UNMODIFIED LSC

Outputs		Input States				
Z_1	Z_2	I_1	I_2	I_3	I_4	I_5
$\underline{1}$	$\underline{0}$	(1)	2		3	
1	$\underline{1}$		(2)	4		
$\underline{0}$	$\underline{1}$	6			(3)	5
$\underline{0}$	$\underline{0}$		7	(4)		5
$\underline{1}$	$\underline{0}$	6			3	(5)
$\underline{1}$	$\underline{1}$	(6)	8		9	
0	$\underline{1}$	1	(7)			
1	1		(8)	4		
$\underline{1}$	$\underline{0}$	1			(9)	10
$\underline{0}$	0			4	9	(10)

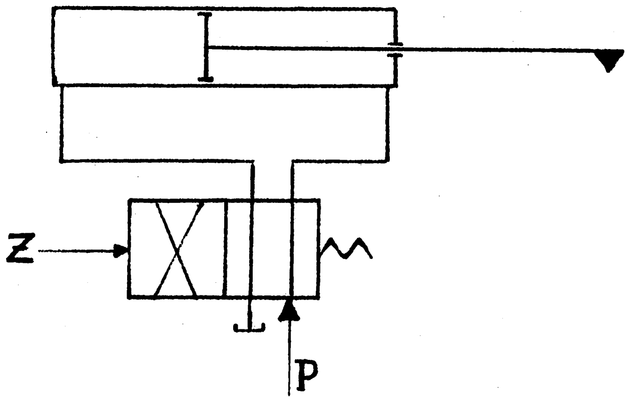


Figure 13. The Assigned Output Circuit

a as there are "1" outputs at States 2, 7, and 8; and therefore, further assessment of these outputs is necessary.

Evaluation of Output Z_1 shows that all output conditions are passive during the times the system is at input state I_2; and hence, a detented output element is assigned to the Z_1 output circuit. Output Z_2 exhibits the other situation -- "1" outputs are found in all states of Column I_2 -- which requires the implementation of a normally extending output circuit for Z_2. These selected output circuit configurations are shown in Figure 14.

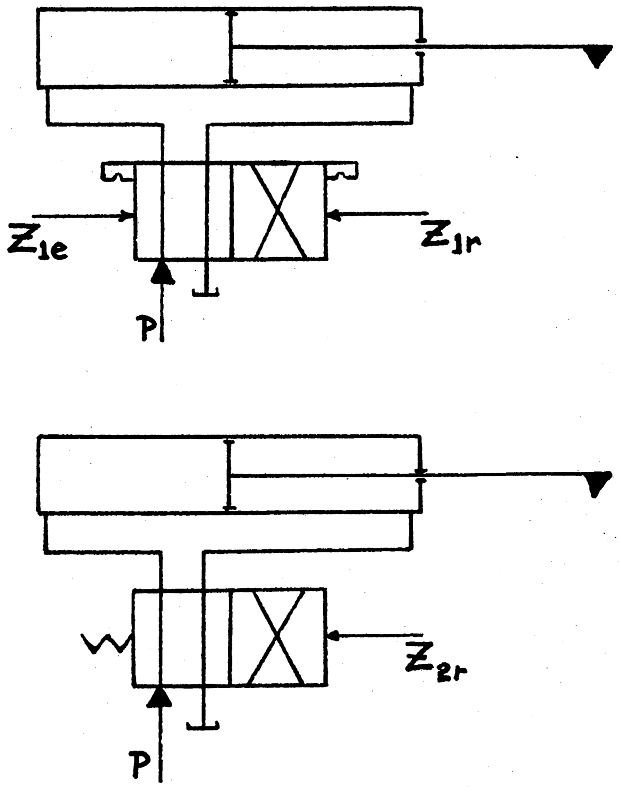


Figure 14. The Selected Output Circuits for Z_1 and Z_2

After the selected output circuits are implemented, it is necessary that the LSC be modified in order to incorporate the new circuit outputs, Z_{1e}, Z_{1r}, and Z_{2r}. This modified LSC can be observed in Table XIV.

TABLE XIV

THE MODIFIED LSC

Outputs			Input States				
Z_{1e}	Z_{1r}	Z_{2r}	I_1	I_2	I_3	I_4	I_5
1	0	1	(1)	2		3	
-	-	0		(2)	4		
0	1	0	6			(3)	5
0	1	1		7	(4)		5
1	0	1	6			3	(5)
1	0	0	(6)	8		9	
-	-	0	1	(7)			
-	-	0		(8)	4		
1	0	1	1			(9)	10
0	1	1			4	9	(10)

Observation of this modified LSC reveals that the circuit outputs Z_{1e}, Z_{1r}, and Z_{2r} satisfy the input eliminating Condition a. Therefore, Column I_2 can be eliminated; resulting in the Reduced LSC shown in Table XV. It can be agreed that the ability to modify the LSC provides additional possibilities for the simplification of the logic specification.

TABLE XV

THE REDUCED LSC

Outputs			Input States			
Z_{1e}	Z_{1r}	Z_{2r}	I_1	I_3	I_4	I_5
1	0	1	(1)	4	3	
0	1	0	6		(3)	5
0	1	1	1	(4)		5
1	0	1	6		3	(5)
1	0	0	(6)	4	9	
1	0	1	1		(9)	10
0	1	1		4	9	(10)

6.4.2. Redundant State Elimination

Redundant states are states which have been represented by other states; either directly or indirectly. Therefore, the presence of such states would unnecessarily complicate the network description. In addition, a superfluous network specification would lead towards the development of more complex circuits, as each unnecessary state would require an individual representation by the implemented network.

There are two types of redundancies which can occur in a logic specification:

1. Duplication

2. Obviation

Duplication is a situation where a machine state, or a group of machine states are represented more than once in the logic specification. Obviation reflects a more complex situation; in which a state, or a group of states are indirectly represented by other machine states. For example, States 1 and 5 of Table XVI are duplicates as they have identical outputs and next states. It is, therefore, concluded that State 5 is State 1 and, hence, State 5 can be eliminated without changing the logic of the specification.

A different situation occurs between States 3 and 8. This situation reflects an "obviation" condition as a new State 9 can be constructed as such that it obviates both States 3 and 8. This new state is tabulated at the bottom part of Table XVI.

At this point, it is important to conclude that the elimination of redundancies resulting from duplication would not change the next state representation of the retained state; while the combination of states in the obviation process would generally increase the number of the next state entries. Relating this fact to deterministic sequential networks, it can be agreed that the process of obviation would transform such networks into stochastic type networks. Although minimal memory machines are obtained by the process, the additional complexity created by such stochastic systems would result in additional costs in the implementation of the network (20, 26). Therefore, the redundant state elimination procedure discussed in this presentation would distinguish two types of state reduction schemes--the redundant state elimination for deterministic networks and the one for stochastic networks.

TABLE XVI

LSC SHOWING DUPLICATION AND OBVIATION OF STATES

Outputs		Input States			
Z_1	Z_2	I_1	I_2	I_3	I_4
0	0	(1)	2		
0	1	1	(2)	4	
1	1			6	(3)
1	0		7	(4)	3
0	0	(5)	2		
1	0		7	(6)	8
0	1	5	(7)	6	
1	-	5			(8)
1	1	5		6	(9)

6.4.3 The Redundant State Elimination for Deterministic Networks

The redundant state elimination for deterministic networks involves the detection of the duplication of states. In order to provide a basis for further discussion of the subject, the term "Duplicative State Equivalency" is defined as follows:

Two machine states can be classified as "Duplicative Equivalent" if the following conditions exist:

A. The two states have the same input state

B. The outputs are either identical, or, where disagreement occurs, don't cares ("-") are involved.

C. They have either the same or duplicative equivalent next states.

Condition C shows that a duplicative equivalency may depend upon the equivalency of other states. In order to tackle this problem in a most efficient manner, the Equivalent Pairs Chart (EPC), which has been widely used in the Classical Synthesis Approach, will be used. An EPC is basically a group of cells, constructed in a matrix form. Each cell in the matrix represents the equivalency of its coordinates; and by inserting an "X" in the cell, a non-equivalent condition is given. One or more pairs of states in the cell indicates a conditional equivalency; while an empty cell means an unconditional equivalency. A conditional equivalency requires that the pairs of states indicated in the cell be equivalent in order that the pair of states under consideration can be classified as equivalent. If such equivalency is not achieved, an "X" is entered in the respective location.

As an illustration, consider the LSC of a deterministic problem shown in Table XVII. Condition A states that a duplicative equivalency between two states can only occur if they have the same input; and therefore the construction of one EPC for each LSC-column would be most appropriate. Observation on the LSC reveals that the duplicative equivalency of States 1 and 3 depends upon the equivalence of States 2 and 4; and this latter pair of states is entered in the first cube, first column of the EPC of Column I_1 (see Figure 15). Furthermore, as States 1 and 7 has different outputs, an X is inserted at location (1, 7).

Following the completion of the EPC, all conditional equivalencies are observed whether or not they depend upon a non-equivalent pair of states. If this is true, the conditional equivalency becomes a non-equivalency and an X is entered in the respective location. For example, the conditional equivalency between States 1 and 9 depends upon the equivalence of States 2 and 10, which happen to be non-equivalent. An X is therefore inserted in location (1, 9) to indicate the non-equivalency of these two states. The finalized EPC's can be observed in Figure 15; where each remaining un-X-ed position indicates the duplicative equivalency between its two candidates.

TABLE XVII

LSC FOR A DETERMINISTIC PROBLEM

Outputs		Input States			
Z_1	Z_2	I_1	I_2	I_3	I_4
0	1	(1)	2		
0	0	3	(2)		
0	1	(3)	4		
1	1		(4)	5	
1	0			(5)	6
0	0	7			(6)
1	1	(7)	8		
0	1	9	(8)		
0	1	(9)	10		
1	1		(10)	11	
1	0			(11)	12
0	0	13			(12)
–	1	(13)	14		
0	–	3	(14)		

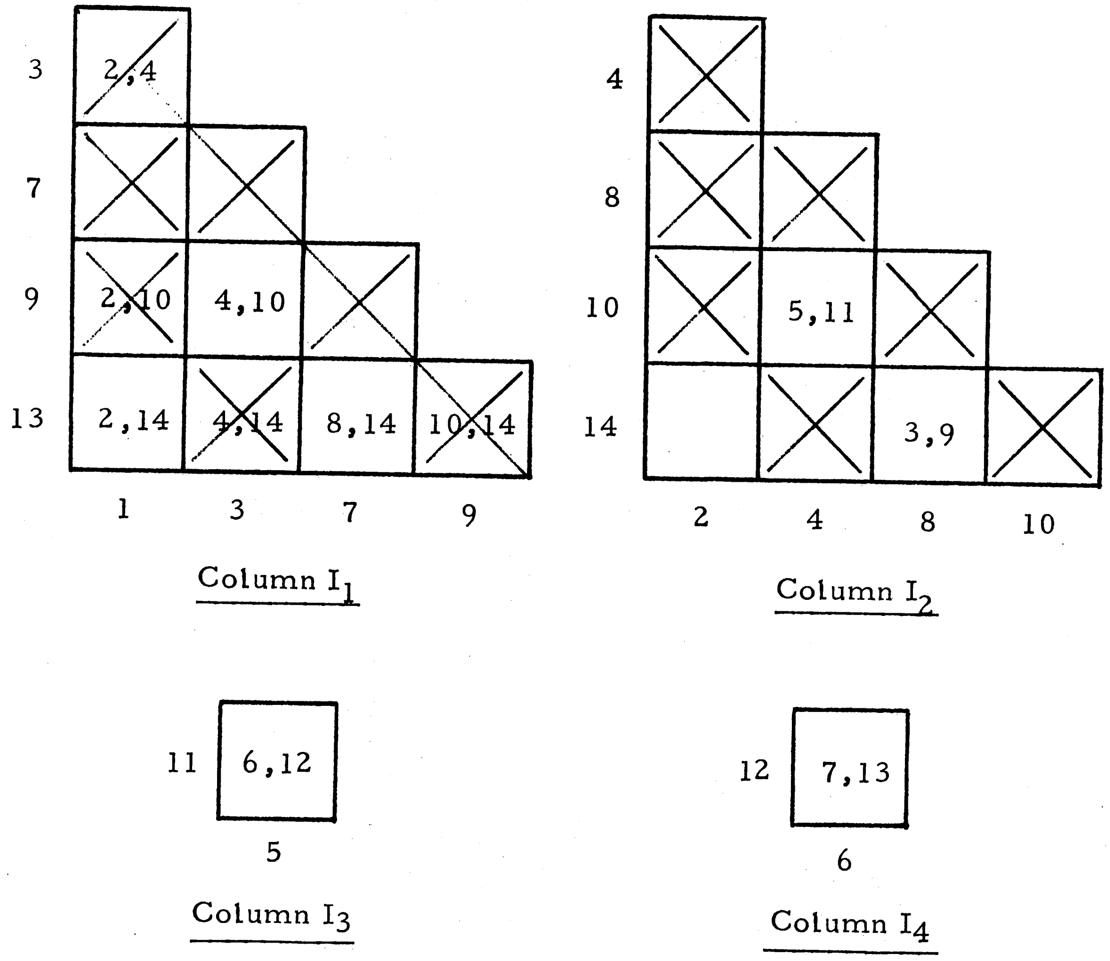


Figure 15. The Equivalent Pairs Chart

6.4.4 The Substitution of Modified Machine States

Once the EPC's are completed, the equivalent states must be combined in order to form a minimal state network. In this step, an attempt is made to group the largest possible number of states to form new states; followed by the careful selection of these new states to form the minimal state machine. These groups, containing maximal combinations of states which can co-exist together to form new machine states, are defined as the "maximal equivalent sets" of states.

There are many approaches available which can be utilized for deriving these maximal equivalent sets. However, for extremely large problems, the partitioning method advanced by A. Grasselli and F. Luccio (16, 17, 18) is considered to be most appropriate and is given in tabular form in this presentation. The method can be performed using the following steps:

1. Form an N-column array, each column being related to a state of the N-state EPC. For the convenience of this discussion, let the columns of the array relate to the columns of the EPC (with the exception of the last column of the array which is not represented in the EPC).
2. Start the iteration process by entering "1"s in all columns of Row 1.
3. Consider the first column of the EPC.
4. Partition every row represented in the array into two rows--one containing all entries except the one related to the column under consideration; while the other containing the excluded entry and all entries which relate to the "un-X-ed" cells of the EPC-column

under consideration. Dashes are inserted in locations where entires are being excluded.

5. Eliminate the row being partitioned.
6. Compare each newly generated row to the other rows and eliminate the ones that are totally contained in the other.
7. Consider the next EPC column and perform Steps 3-7 until all EPC columns have been assessed.
8. Each of the remaining rows reflects a maximal equivalent set.

In order to illustrate this procedure, consider the first EPC of Figure 15. As there are 5 states involved in this EPC, a five column array is formed. Initially, the array is a 1-row array as shown in Row 1 of Table XVIII. Considering the first column of the EPC, Row 1 can be partitioned into two rows as illustrated by Rows 2 and 3. Note that Row 2 does not contain the first entry while Row 3 contains this first entry and the entry relating to State 13, which is the only un-X-ed entry in the first EPC-column. Following the generation of these two rows, Row 1 is eliminated; which is shown as a checkmark in Table XVIII.

The next step involves the partioning of Rows 2 and 3 with respect to the second EPC-column. The partioning of Row 2 results in the generation of Rows 4 and 5, which are the group that excludes State 3 and the group that contains State 3, respectively. Row 3 exhibits a different situation--the absence of State 3 in its representation causes this row to be unchanged during the second stage of this process. Progressing through the iteration, it can be seen in Table XVIII that Rows 3, 7, and 9 remain unchecked and, hence, they represent the maximal equivalent sets of the first LSC column.

TABLE XVIII

PARTITIONING TABLE

Row No.	States					
	1	3	7	9	13	
1	1	1	1	1	1	✓
2	–	1	1	1	1	✓
3	1	–	–	–	1	
4	–	–	1	1	1	✓
5	–	1	–	1	–	✓
6	–	–	–	1	1	✓
7	–	–	1	–	1	
8	–	1	–	–	–	✓
9	–	1	–	1	–	
10	–	–	–	–	1	✓
11	–	–	–	1	–	✓

After all columns of the LSC has been considered, the next step is to select a minimal combination of maximal equivalent sets for representing each state of the chart.

This can be performed by means of available accounting techniques (e.g., the method by W. V. Quine (31). For the previous example, it can be shown that all equivalent sets must be drafted in order to represent all states of the LSC.

6.4.5 The Formulation of the Reduced Logic Specification

Having selected the minimal number of equivalent sets to cover the states of the logic specification, an attempt is made towards the construction of a new compacted logic specification, which is referred to as the "Reduced Logic Specification" (RLS). When the LSC format is used, the term "Reduced Specification Chart" (RSC) is commonplace. The main objective of this attempt is to utilize the selected equivalent sets and to project their output and next states from the logic specification onto the RLS.

The approach is initiated by identifying the "Next State Sets" which result from the grouping of states in the maximal equivalent sets. A next state set is a group of states which are the next states of states contained in an equivalent set, and which have the same input state. For example, the next state set of the equivalent set (1, 13) is (2, 14) (see Table XVII). Similarly, the next state set of Set (6, 12) is (7, 13). Note that in deterministic cases there are only one next state set for each equivalent set; which may not be true for stochastic problems.

After the next state sets of each equivalent set have been identified, the formulation of the RLS can be performed using the following steps:

1. Identify each selected equivalent set by using new state numbers. Assign each new state to a row of the new logic specification.
2. In the first column of the RLS, list the combined outputs of the selected sets in the respective rows. The combination of the outputs can be obtained by the *-product of all output states represented in each set.
3. The logic interaction between the machine states is formed by identifying each next state set using a selected set which fully covers the next state set under consideration. If no such selected sets exist which can represent one or more next state sets, use the least combination of maximal equivalent sets in order to satisfy these next state requirements.

In order to illustrate the formulation of the RLS, the example problem represented in Table XVII is considered. As the problem has been given in an LSC format, the retaining of this format results in the development of the RSC. By assigning the maximal equivalent sets (1, 13), (2, 14), (3, 9), (4, 10), (5, 11), (6, 12), (7, 13) and (8.14) with the new state numbers (1', 2', 3', ..., 8'), respectively, the RSC can be constructed as shown in Table XIX. Note for example that the next state set of "New" State 7 '(originally

States 7, 13) is the set (8, 14); which is covered by the "New" State 8'.

TABLE XIX

THE REDUCED SPECIFICATION CHART

Outputs		Input States			
Z_1	Z_2	I_1	I_2	I_3	I_4
0	1	(1')	2'		
0	0	3'	(2')		
0	1	(3')	4'		
1	1		(4')	5'	
1	0			(5')	6'
0	0	7'			(6')
1	1	(7')	8'		
0	1	3'	(8')		

It can be agreed that a major reduction of the problem description has been achieved. At this moment it is important to point out that the deterministic condition of the problem has been retained in the RSC.

6.4.6 The Redundant State Elimination for Stochastic Networks

Basically, the process of reducing a stochastic system is identical to that of a deterministic network, with one exception: that is the criterion of the equivalency. When dealing with deterministic networks, the process of combining the states is limited to the duplication of states. Such an approach was selected in order that the given deterministic problem remain deterministic throughout the synthesis process. However, when the switching problem is stochastic in nature, no limitation whatsoever is imposed as to which states can or cannot coexist together to form the new state. Therefore, the main objective of this approach should be the optimal reduction of the logic specification.

Therefore, the formulation of the equivalency criterion can be based upon both the obviation and duplication of machine states, which results in the following definition of the machine state equivalency:

Two machine states can be classified as equivalent if the following conditions are met:

A. The two states are represented by the same input state.

B. The outputs are either identical, or, where disagreement occurs, don't cares ("-") are involved.

C. They have either the same, or equivalent states; or, when disagreement prevails, it should involve states with different input states.

In order to illustrate the reduction approach, the LSC shown in Table XX is considered. The EPC's of this LSC can be constructed as shown in Figure 16. Note that States 2

TABLE XX

LSC FOR THE STOCHASTIC PROBLEM

Outputs		Input States			
Z_1	Z_2	I_1	I_2	I_3	I_4
0	0	(1)	2	3	6
0	1	5	(2)		10
-	1	1		(3)	10
0	1		(4)	8	9
0	0	(5)	4	7	10
1	1		4	3	(6)
0	1	5		(7)	6
1	1	1		(8)	9
1	0		2	3	(9)
1	-		2	7	(10)

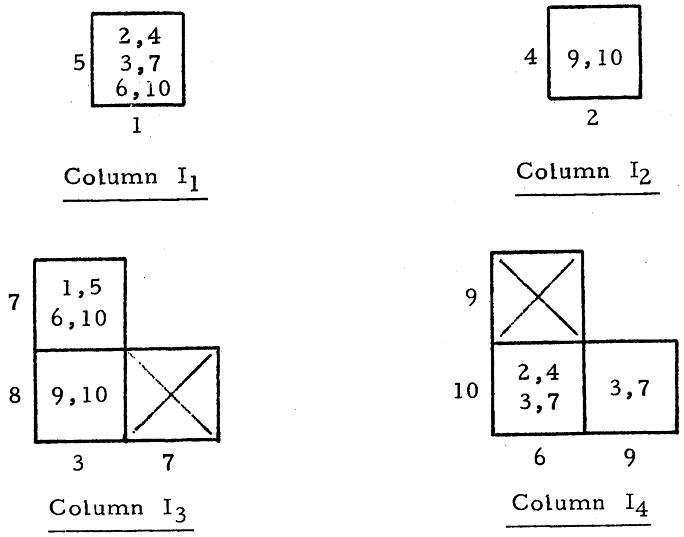


Figure 16. The EPC for the Stochastic Problem

and 4 are not equivalent when a duplicative criterion is used.

The remaining steps in the reduction process is identical to the process conducted in solving the deterministic case. It can be shown, that the maximal equivalent sets of the LSC is as follows:

(1, 5), (2, 4), (3, 7), (3, 8), (6, 10), (9, 10).

Again, it can be observed that all the above maximal equivalent sets are essential for the coverage of all states of the machine. Numbering these sets consecutively by 1', 2', ..., 6', the RSC can be constructed as shown in Table XXI.

TABLE XXI

RSC FOR THE STOCHASTIC PROBLEM

Outputs		Input States			
Z_1	Z_2	I_1	I_2	I_3	I_4
0	0	(1')	2'	3'	5'
0	1	1'	(2')	4'	6'
0	1	1'		(3')	5'
1	1	1'		(4')	6'
1	1		2'	3'	(5')
1	0		2'	3'	(6')

6.5 The Assignment of Memories

A sequential network is characterized by the presence of memory functions--an important ingredient which enable the network to record the history of the past machine states. There are two ways a memory function can be satisfied; and these are:

1. The application of available output variables to perform the memory functions;
2. The implementation of actual memory elements in the circuit.

It is realized that the utilization of available output variables as memory variables would be desirable as they do not require additional implementation for the "updating" (or SET/RESET) of the memory functions. Unfortunately, not all output variables can be used effectively for performing the specified recording task. Moreover, available outputs may not suffice the memory requirements of the network specification. Therefore, actual memory elements may still be required for complementing this task.

There are various conditions which must be followed in order that this "output feedback" approach can be successful. For instance, the state of an output variable can be effectively used as a memory state of a machine state only if the output is passive in the particular machine state. The utilization of active outputs would lead to the generation of races, and therefore, it will not be condoned in this presentation. Furthermore, as the pattern of the output actuation does not have any relationships which correspond to the necessity for memory states, a utilization of an output as a memory variable may not always result in the reduction of the memory elements required by the network. Under such circumstances, it can be agreed that the output under consideration should not be selected for performing the particular memory function.

The assignment of the "actual" memories and the "output feedback" memories is performed in a similar manner as in the "non-classical" synthesis approaches, where each input state is governed by an individual set of memory variables. Such an approach would have permit the utilization of each memory element as passive elements--a feature which would reduce the number of implemented elements by at least one AND element.

The output assigning approach can be outlined as follows:

1. For each input state, count the number of states in the logic specification. The number of necessary memories for each input state is represented by the larger integer value of $\log_2(N_s)$, where N_s is the number of states having the particular input state.
2. Select one output variable. The pairs of outputs which are representing detented power elements are not considered individually but are replaced by their "intended outputs".

3. Select one input state, which has not yet been assigned an output feedback variable.
4. Partition the set of states represented by this input state into two groups--the one which has passive zero outputs and the one with passive one outputs. Include all states which do not satisfy these two partitioning conditions into both groups.
5. Count the number of states of the larger group of states ($=N_1$). Determine the value of $\log_2(N_1)$, which represents the number of actual memory elements that are still required for the particular input state. If the larger integer of this value is less than the value obtained previously in Step 1, then, the application of the output under consideration is successful. If not, disregard the results, select the next input state which has not yet been assigned an output feedback variable and return to Step 4. Perform Steps 2 to 5 iteratively until all outputs have been considered.

After the output feedback variables are selected and applied, memory elements are assigned to the remaining non-unique states. The augmentation of these memories should be performed such that minimal SET-ing and RESET-ing operations occur during the machine cycle.

In order to illustrate this output feedback and memory augmentation procedure, consider the example problem given in Table XXII. In this table, the active outputs are underlined as shown in the first two columns of the table. By counting the number of states in each of the columns of the LSC, it is realized that one memory element is required for each input state. Partitioning the machine states of Column I_1 with respect to Output Z_1 results in two states containing one state, which does not require the augmentation of actual memory devices. This shows that Output Z_1 can be used as a memory variable in connection to Input State I_1. This information is recorded by listing the passive states which are utilized as the memory states in the "memory assignment" subtable (see Table XXII).

TABLE XXII

THE RSC WITH MEMORY ASSIGNMENTS

Outputs		Input States				Memory Assignment			
Z_1	Z_2	I_1	I_2	I_3	I_4	Z_1	Z_2	Y_1	Y_2
0	1	(1)	2		3	0		0 R	0 R
$\underline{1}$	1	4	(2)	5				0	-
$\underline{1}$	$\underline{0}$	4		6	(3)			-	0
1	$\underline{0}$	(4)	7		8	1		1 S	1 S
$\underline{0}$	1		2	(5)	8		1	0 R	1 S
1	0		7	(6)	3		0	1 S	0
$\underline{0}$	$\underline{1}$	1	(7)	5				1	-
$\underline{1}$	$\underline{1}$	4		5	(8)			-	1

Partioning Column I_2 with respect to Z_2 proves to be unsuccessful: The partitioned sets are Sets (2, 7) and (7), which still requires the utilization of a memory; and, therefore, Z_2 cannot serve as a memory variable for Input State I_2. Further observation on Table XXII reveals that Output Z_2 can perform a successful partitioned between States 5 and 6 (Column I_3).

As Columns I_1 and I_3 have been "memory augmented" using the feedback outputs, only Columns I_2 and I_4 need to be considered further for additional memory elements. The augmentation of the memory elements is performed by assigning a different memory state for each non-unique machine state. In order that a minimal number of SET-ing and RESET-ing takes place, an assignment in a Gray code form as suggested by G. E. Maroney (21) is most appropriate. In the example problem of Table XXII, it has been established that each column requires only one memory element, and therefore, a "0" is assigned to one non-unique state while the other state is assigned a "1". The memory augmentation can be observed in the Y_1 and Y_2-columns where "0"s are assigned to State 2 in Column Y_1 and State 3 in Column Y_2; and "1"s are placed at States 7 and 8 in Columns Y_1 and Y_2, respectively.

After the "0" and "1" states of the memory elements have been determined, additional outputs are to be considered. These outputs are commonly known as the "secondary outputs" of the sequential system. Secondary outputs are output signals which are utilized for actuating the memory elements. Basically, there are two actuating signals which should be considered; namely, the SET and the RESET signals. The SET state must be achieved in the machine states previous to the states where the "1" memory state is required. Similarly, the RESET state must occur prior to the states where a "0" state is desired.

The assignment of the SET and RESET states are most conveniently determined by the following steps:

1. For each augmented "0" condition, assign "0" to all its previous states. Similarly, "1"s are assigned to all previous states of an augmented "1" condition.
2. For each memory element, observe all "0"s and "1"s of the table and substitute R's for "0"s and S's for "1"s if they have a different previous state.

As an illustration, in the previous example, "0"s are placed in the Y_1 column at States 1 and 5 as these states are the previous states of State 2. Examining States 1 and 5 reveals that the previous condition of Y_1 is not a "0" and therefore "R's are inserted to indicate RESET states. The completed memory assignment can be observed in Table XXII.

Actually, memory actuation can be performed at any state provided that the intended memory state is achieved. In relation to this, a SET and RESET state can be replaced by all its previous states if conditions permit. For example, in Column Y_2 of the previous example, the SET signal of State 5 can be replaced by SET signals at States 2, 7, and 8. However, in stochastic type problems (as in the previous example) such substitution would generally increase the complexity of the secondary output representations. This is caused by the fact that in stochastic problems, the machine states may have multiple previous states.

On the other hand, deterministic problems will benefit from such signal substitution schemes. A selection can be made as where a SET or RESET should occur in order to avoid the generation of seldomly used signals. Oftentimes, a proper selection may lead towards a major reduction of the network implementation.

Basically, the actuation signal selection approach attempts to distinguish between states which have been used for generating an output (either primary or secondary) and states which are never used previously. The approach also stresses the utilization of unaugmented machine states in order to avoid the utilization of unessential memory devices. The selection criterion can be outlined as follows:

1. Isolate the machine states which represent solely "0" or "-" circuit outputs. Classify these states as "unessential" states, while the remaining states are termed as essential. It should be noted that for detented output elements, unessential states relate to the passive states of the outputs.

2. Reclassify as essential all unessential states which are connected to essential memory elements. Essential memory elements are memory elements which have been used to represent essential states.
3. Select one essential memory element.
4. Observe the SET state and inspect whether or not it occurs in an augmented state. If so, try to relocate the SET state by successively moving it to a previous state until either an unaugmented essential state is achieved or a "0" state prohibits further relocations of the SET state. When no such unaugmented essential state is found, try to locate an unaugmented (unessential) state in a similar manner. If the relocating scheme is still unsuccessful, the final attempt is to locate an essential augmented state. When all the above efforts are fruitless, return the SET state to its original state and classify this state and all states that are connected to the augmented memories as essential.
5. Observe the RESET state and perform relocations as in Step 4. It should be noted that a "1" state would prohibit further relocation of a RESET state.
6. Enter "1"s in all states occurring between a

relocated SET state and the required "1" location. Similarly, "0"s are inserted in all locations between the relocated RESET and the desired "0" states.

7. Perform Steps 3 through 6 iteratively until no essential memories exist. The remaining unessential memories are eliminated from the table.

In order to illustrate this procedure, consider the RSC for the deterministic problem as shown in Table XXIII. Note that the outputs Z_1 and Z_2 have been used as memory variables for columns I_2 and I_3, respectively.

TABLE XXIII

RSC FOR A DETERMINISTIC PROBLEM AND ITS MEMORY AUGMENTATION

Outputs		Input States				Memory Assignment			
Z_1	Z_2	I_1	I_2	I_3	I_4	Z_1	Z_2	Y_1	Y_2
$\underline{0}$	$\underline{0}$	(1)	2					R	R
0	0		(2)	3		0		0	0
$\underline{1}$	0		4	(3)			0	0	0
1	0		(4)	5		1		0	S
1	0			(5)	6		0		1
1	$\underline{1}$			7	(6)			S	
1	1		8	(7)			1	1	
1	1	1	(8)			1		1	

The essential states of this table are States 1, 3, and 6; and therefore, at this moment, only Memory Y_2 is classified as essential. Furthermore, according to Step 2, State 5 is classified as essential due to its association with Y_2. Observation of Y_2 reveals that the SET state cannot be relocated as it is encompassed by the "0" and "1" required states of Y_2. The RESET state shows a different situation. It occurs in an augmented State 2; and a relocation can be made to an unaugmented essential State 1.

Finally, the utilization of State 4 by Y_2 causes Y_1 to be essential and, therefore, its SET and RESET should also be considered. The relocated actuating signals can be observed in Table XXIII.

6.6 The Formulation of the Network Equations

Having reached this stage, the sequential part of the synthesis has actually been completed. The remaining part of the synthesis is pure combinational, as it involves only the interpretation of the synthesized charts and simplifying them in order to obtain the desired, minimal configuration.

6.6.1 The Simplification of the Input States

The simplification of the input states is a necessary intermediate step towards the simplification of the network representation. Such a step is especially useful when multiple outputs (primary or secondary) are present, which is a characteristic possessed by sequential systems.

The simplification process can be most conveniently pursued using the &-product which has been discussed in Chapter V. The process is initiated by tabulating all input states including the input states which have been eliminated in the previous simplification steps. The &-product is then performed upon each input state with respect to all other states. As an example, consider the Input State Simplification Table shown in Table XXIV, in which the input states are presented in a numerical form.

The simplification of the first input state, for example, is achieved by the following product:

$$A = (1\ 0\ 0\ 0\ 0\ 0)\ \&\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} = (1\ 0\ -\ 0\ -\ -) \qquad (6\text{-}5)$$

The remaining input states are simplified in a similar manner. Note that the final input state of Table XXIV has been eliminated previously, and is therefore not simplified.

TABLE XXIV

THE INPUT STATE SIMPLIFICATION TABLE

Original Input States						Simplified Input States					
a	b	c	d	e	f	a	b	c	d	e	f
1	0	0	0	0	0	1	0	-	0	-	-
0	1	0	0	0	0	0	1	-	-	-	-
1	1	0	0	0	0	1	1	-	-	-	-
1	0	0	1	0	0	1	0	-	1	-	-
0	0	0	1	0	0	0	-	0	1	-	-
0	0	1	1	0	0	-	-	1	1	-	-
0	0	1	0	0	0	-	-	1	0	-	-
0	0	0	0	1	0	-	-	-	-	1	-
0	0	0	0	0	1	-	-	-	-	-	1
*0	0	0	0	0	0						

*eliminated in previous steps.

6.6.2 The Interpretation of the Synthesis Results

After the input states have been individually simplified, the interpretation of the synthesis results is quite straightforward. The representation of each machine state is simply the combination of the input state and its memory augmentation.

It should be noted, that there are various types of outputs which should be considered. These are:

1. The primary outputs which relates to spring return output elements or other "single" output representations.
2. The primary outputs which relate to detented output elements.
3. The secondary outputs.

Spring return output elements would duplicate the actuating signal and, therefore, the output representations of such elements are derived by equating the "1"s of the desired output. All "-"s and unspecified states can be considered as don't cares.

The detented outputs have two circuit output representations for each intended output, which are opposing each other. Depending upon their particular function, they are often denoted as SET/RESET signals or as EXTEND/RETRACT signals. The circuit output representation for the SET signal is obtained by replacing all passive "1"s of the intended output by "-"s, while the representation for the RESET signal is derived by replacing all active "0"s by "1"s, all passive "0" by "-"s, and all "1"s by "0"s. Using this substitution approach, no conflicting outputs can occur. The secondary outputs are treated in the same manner as the detented outputs; by assuming the "S"s as active "1"s and the "R"s as active "0"s.

It is realized, that outputs represented by more than one machine state, or outputs which includes some don't care states may be further simplified. However, due to the presence of other outputs (either primary or secondary) a simplification which includes the further reduction of the input states is useless; it may even complicate the implementation of the network equation. Simplification should, therefore, be performed in relation to the memory elements, and this is best realized by simplifying the representation in parts, where each part possesses the same input state. For example, considering the synthesized chart of Table XXII, the representation of th Z_1-SET signal is:

$$Z_1 = +(\circ(I_2,Y_1'),\cdot(I_4, Y_2'),\cdot(I_4,Y_2)) \qquad (6\text{-}6)$$

As all unspecified states and the "1" states are considered as don't cares, the complementation approach would be most appropriate. Hence, the complement is derived as follows:

$$Z_1' = +(\circ(I_1,Z_1'),\circ(I_2,Y_1),\cdot(I_3,Z_2))$$

$$= +(\circ(I_1,Z_1'),\circ(I_2,Y_1),\cdot(I_3,Z_2),\cdot(I_4,0)) \qquad (6\text{-}7)$$

Or, representation of Z_1' relative to each input state results in the following representations for input states I_2 and I_4:

$$Z_1'{}_{I_2} = \circ(I_2,Y_1) \qquad (6\text{-}8a)$$

$$Z_1'{}_{I_4} = \circ(I_4,0) \qquad (6\text{-}8b)$$

Complementing the above expressions while confining the complement in the respective columns, results in the following:

$$Z_{1_{I_2}} = .(+I_2,+(I_2',Y_1')) = .(I_2, Y_1') \qquad (6\text{-}9a)$$

$$Z_{1_{I_4}} = .(+I_4,+(I_4',1)) = .(I_4, 1) \qquad (6\text{-}9b)$$

The Z_1 expression is, therefore, simplified as follows:

$$Z_1 = +(.(I_2,Y_1'),.(I_4)) \qquad (6\text{-}10)$$

It should be noted at this time that the direct complementation of the Z_1' expression of Equation (6-7) may not be successful as each input state may have been optimally simplified.

The interpretation of the synthesis results concludes the presentation of the sequential logic synthesis, which has been thoroughly discussed in this chapter. Simple example problems have been used throughout the discussion in order to provide a clear insight into the procedure.

6.7 Procedure Outline

The necessary steps and operations required for synthesizing asynchronous sequential networks have been compiled and presented in this chapter. In order to aid the designer in the utilization of the operations in a most efficient manner, the synthesis method is summarized in a step-by-step outline as follows:

1. Develop the logic specification from the

problem description.

2. Recognize the type of the problem. Determine whether it is a combinational, deterministic sequential, or a stochastic sequential network.
3. Simplify the input states by using the "&-product". This step can actually be performed at any stage in the synthesis, without altering the final results.
4. Eliminate the uninfluential input states, if any. When this step is performed, the utilization of the selected output elements is mandatory.
5. Eliminate the redundant states in the system.
6. Determine the active and passive outputs.
7. When Step 4 is unsuccessful, determine the most appropriate output elements for each output variable.
8. Try to incorporate the output variables as memory variables of the system. An output variable should not be used for representing more than one input state.
9. Assign the remaining unaugmented states by actual memory variables.
10. Assign SET and RESET signals to each of the memory elements.
11. Derive the primary and secondary output equations. Utilize the complementation approach

for the simplification of the memory states.

12. Implement the network equations.

CHAPTER VII

VERIFICATION OF THE METHOD

7.1 General

The combinational and sequential synthesis approaches discussed in this presentation offers a powerful means for assessing complex fluid logic problems. It is also claimed that the method offers near minimal simplification of the network implementation.

It is the purpose of this chapter to properly justify these claims and show the effectiveness of the method relative to existing synthesis methods.

7.2 Comparisons to Other Techniques

It is realized that no mathematical proof exists which can be used for verifying the capability of a synthesis method for producing minimal networks. This is caused by the random behavior of the logic problem. Therefore, the only means for verifying the minimality of a synthesis result is by comparing it to the results of other existing synthesis methods. For the purpose of this discussion, three known synthesis techniques were selected, which are the Classical, the Change Signal (20) and the Total Signal methods (21). Eight problems of different types are resolved using each of these techniques, and are implemented with available logic elements. These eight problems are presented in the Appendix of this presentation.

After each result was obtained and implemented, the number of logic elements used were counted, and they are tabulated in Table XXV. Compilation of the results show that the method advanced in this presentation produces less complex networks than the ones resulting from other synthesis methods. Although the degree of reduction varies from one problem to the other, no case of network expansion was encountered.

TABLE XXV

COMPARISONS OF THE IMPLEMENTED RESULTS

Problem No.:	Number of Implemented Elements				Reduction Relative to:		
	New Method	Classical Method	Change Signal Method	Total Signal Method	Classical Method	Change Signal Method	Total Signal Method
1	11	15	*	13	26.6%	-	15.7%
2	19	27	*	22	29.6%	-	13.6%
3	11	15	13	20	26.6%	15.7%	45.0%
4	10	13	*	25	25.7%	-	60.0%
5	11	30	*	13	63.3%	-	15.7%
6	0	6	6	8	100.0%	100.0%	100.0%
7	7	16	9	7	56.2%	22.2%	0.0%
8	12	41	12	36	70.7%	0.0%	66.6%

* Not applicable.

7.3 Computer Programming

Undoubtedly, the existence of computerized synthesis programs would be most valuable to designers who want to design large scale fluid logic systems. The use of digital computers would most likely avoid the creation of "common human errors" during the synthesis process of the network. Such errors tend to occur during an enduring process of manual synthesis. It is, therefore, realized that computer programming should be directed towards the synthesis of large scale networks. Note that a computer program capable of resolving only small networks is worthless; as such small networks are most conveniently resolved manually.

Therefore, a synthesis method can be classified as suitable for large scale systems only if it is developed as such that it can be computerized. Moreover, the capacity of the computer program relative to the computer core utilization and execution time would reflect the effectiveness of the synthesis algorithm.

While keeping the above arguments in mind, a computer aided synthesis program based upon the technique presented in this thesis has been designed using the FORTRAN IV programming language. The program uses the Sequence Matrix format for performing the manipulations necessary in the synthesis, as this format occupies the minimal amount of computer core. Although the program is currently not yet in its refined stage, the ability of the program for resolving extremely large problems is evident. At present, the program has been designed to resolve problems which have up to 200 input variables, 200 outputs, 200 memories, and 200 states. The computer core requirements for executing the program is relatively small (140 Kbytes), which shows the effectiveness of the program.

CHAPTER VIII

SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS

8.1 Summary

A procedure which is capable of deriving complex asynchronous sequential logic networks has been developed. The presentation has been divided into three major parts—the algebra, the combinational logic synthesis, and finally, the sequential logic synthesis.

The presentation commences with the discussion of a new, generalized algebra. A universal set of operators is used in the algebra in order to provide a convenient means for assessing various types of logical operators. Next, the various combinational synthesis methods are presented for supporting the sequential synthesis approach which was given in the succeeding chapter of this thesis.

The sequential synthesis approach consists of various steps, and these steps were presented according to the five classifications of the operations; which are the formulation of the logic specification, the selection of the peripheral equipment, the simplification of the specification, the assignment of the memories and finally the formulation of the network equations. The presentation is concluded with an outline of the integrated procedure.

8.2 Conclusions

A new insight into the theory of fluid switching circuits has been revealed. The algorithmic type presentation provides convenience in executing the synthesis method formulated herein, especially when digital computers are employed.

The development of the generalized "switching circuit algebra" offers a unique in-depth view of logical expression which utilizes non-standard operators. This generalization also allows convenient assessment of the algebra using digital computers, as the operators of the expression can be considered independently from the desired operations of the algebra. Various algebraic operations may, therefore, be realized without prior knowledge of the actual operators involved.

There are basically three combinational synthesis approaches which have been selected and presented in Chapter V. These three approaches are of great importance for computer aided synthesis, as they are most suited for synthesizing extremely large problems with large numbers of variables.

New directions were also offered in the area of sequential logic synthesis. For the first time, the selection of the peripheral equipment is performed in the synthesis. It is realized that a correct selection of such input--output circuits would result in additional simplification of the implemented network; as the utilization of different output circuits completely alters the specification of the problem.

Furthermore, the method features the utilization of output signals as inputs to the logic circuit. This reduces the need for memory elements and their excitation; and further simplification of the synthesized networks may occur.

Finally, it is concluded that several advancements in the area of Fluid Logic have been realized in this study. The development of the new method will hopefully aid the future designer in solving complex logic problems in a most efficient manner.

8.3 Recommendations for Further Study

After completing this study, the author feels that additional work in this area would be beneficial to future implementations of logic systems. The following points are still lacking as of today and is, therefore, suggested as topics for further study:

1. The further advancement of the "Switching Circuit Algebra". It is realized that the algebra presented earlier in this thesis is far from complete. Additional postulates and/or theorems would be necessary for the effective utilization of the "non-conventional" operators. The study should not exclude the possibilities of developing new, practical operators which satisfy the algebra.

2. The direct application of different operators in logic system synthesis. Up to now the synthesis of sequential networks has been performed in a conventional AND-OR-NOT fashion. The transformation to other operators (such as NOR's and NAND's) was performed in the combinational portion of the synthesis. The author feels that a sequential synthesis method which directly synthesizes the problem using the intended operators would be most practical.
3. The adaptation of the method to synchronous systems. As with all other synthesis techniques, the method advanced in this thesis is applicable to electronic circuits as well. Due to the fact that most electronic logic problems deal with synchronous networks, the modification of the new synthesis method to accept synchronous systems is necessary.
4. The modification of the Classical technique to include "output feedback". The classical Huffman technique has been continuedly used in the past as a "referee" method; and therefore, it should be subsequently improved in order to assess other synthesis approaches.

BIBLIOGRAPHY

(1) Holbrook, E. L. "A Design Approach to Pneumatic Control." Hydraulics and Pneumatics (October 1965).

(2) Kutti, A. K. "O graficheskom izobrazhenii rabochego skhem." Trudy Leningradskoi Experimental'noi Elektrotekhnicheskoi Laboratorii, Vol. 8 (1928), 11-28; Tr. Moore, E. F. "On a Graphical Representation of the Operating Regime of Circuits." Sequential Machines : Selected Papers, 1st. Ed. Mass.: Addison Wesley Pub. Co., 1964, 228-235.

(3) Shannon, C. E. "A Symbolic Analysis of Relay and Switching Circuits." Trans. AIEE, Vol. 57 (1938), 713-723.

(4) Karnaugh, M. "The Map Method for Logic Synthesis of Combinational Logic Circuits." AIEE Trans. on Communications and Electronics, Vol. 72 (1953), 593-599.

(5) Veitch, E. W. "A Chart Method for Simplifying Truth Functions." Proc. Assoc. Computing Machinery (May 2 & 3, 1952), 127-133.

(6) Huffman, D. A. "The Synthesis of Sequential Switching Circuits." Journ. of the Franklin Inst., Vol. 257, No. 3 (March, 1954), 161-190; and No. 4 (April, 1954), 275-303.

(7) Fitch, E. C., Jr. "The Synthesis and Analysis of Fluid Control Networks." (Ph.D. Dissertation, University of Oklahoma, 1964.)

(8) Paull, M. C., and S. H. Unger. "Minimizing the Number of States in Incompletely Specified Sequential Switching Functions." IRE Trans. on Elect. Computers, Vol. EC-8 (1959), 356-367.

(9) Ginsburg, S. "On the Reduction of Superfluous States in a Sequential Machine." Journ. Assoc. Computing Machinery, Vol. 6 (April, 1959), 259-282.

(10) Ginsburg, S. "A Technique for the Reduction of a Given Machine to a Minimal State Machine." IRE Trans. on Elect. Comp., Vol. EC-8 (1959), 346-356.

(11) Ginsburg, S. "Synthesis of Minimal State Machines." IRE Trans on Elect. Comp., Vol. EC-8 (1959), 446-449.

(12) Narashiman, R. "Minimizing Incompletely Specified Sequential Switching Functions." IRE Trans. on Elect. Comp., Vol. EC-10 (1961), 531-532.

(13) Eichelberger, E. B. "Obtaining a Minimum-State Compressed Flow Table." Digital Systems Lab., Princeton University, Technical Report No. 20 (April 1962).

(14) McCluskey, E. J., Jr. "Minimum-State Sequential Circuits for a Restricted Class of Incompletely Specified Flow Tables." Bell System Tech. Journ., Vol. 41, No. 6 (November 1962), 1759-1768.

(15) Marcus, M. P. "Derivation of Maximal Compatibles Using Boolean Algebra." IBM Research & Development Journal, Vol. 8 (November, 1964), 537-538.

(16) Grasselli, A., and F. Luccio. "A Method for Minimizing the Number of Internal States in Incompletely Specified Sequential Networks." IEEE Trans. on Elect. Comp., Vol. EC-14 (1965), 350-359.

(17) Graselli, A. "Minimal Closed Partitions for Incompletely Specified Flow Tables." IEEE Trans. on Elect. Comp., Vol. EC-15 (1966), 245-249.

(18) Luccio, F. "Extending Definition of Prime Compatibility Classes of States in Incomplete Sequential Machine Reduction." IEEE Trans. on Elect. Comp., Vol. C-18 (1969), 537-540.

(19) Biswas, N. N. "State Minimization of Incompletely Specified Sequential Machines." IEEE Trans. on Elect. Comp., Vol. C-23 (1974), 80-84.

(20) Cole, J. H. "Synthesis of Optimum, Complex Fluid Logic Sequential Circuits." (Ph.D. Dissertation, Oklahoma State University, 1968.)

(21) Maroney, G. E. "A Synthesis Technique for Asynchronous Digital Control Networks." (M.S. Report, Oklahoma State University, 1969.)

(22) Maroney, G. E., and E. C. Fitch. "Stochastic Type Digital Fluid Control System Synthesis." Controls and Systems Conference, Chicago, Ill. (May, 1970).

(23) Surjaatmadja, J. B. "A Generalized Method for Synthesizing Optimal Fluid Logic Networks." Annual Fluid Power Research Conference, Oklahoma State University, Report No. R73-FL-3 (1973).

(24) Surjaatmadja, J. B. "Maximizing Decision-Making Capabilities While Minimizing Fluid Logic Hardware." Annual Fluid Power Research Conference, Oklahoma State University, Paper No. P73-FL-5 (1973).

(25) Fitch, E. C., Jr., and J. B. Surjaatmadja. "A Universal Synthesis Method for Fluid Logic Networks." Annual Fluid Power Res. Conf., Oklahoma State University, Paper No. P76-43 (1976).

(26) Chen, R. M. H., and K. Foster. "A Computer Aided Design Method Specially Applicable to Fluidic-Pneumatic Sequential Control Circuits." ASME-WAM, Fluidics Comm., Paper No. 70-WA/Flcs-17 (1970).

(27) Surjaatmadja, J. B. "Arguments for Promoting the Effective Utilization of Memory Logic Hardware." Annual Fluid Power Res. Conf., Oklahoma State University, Paper No. P73-FL-4 (1973).

(28) Surjaatmadja, J. B. "Output to Memory Matching for the Synthesis of Hazard-Free Fluidic Networks." Annual Fluid Power Res. Conf., Oklahoma State University, Paper No. P74-30 (1974).

(29) Surjaatmadja, J. B., and E. C. Fitch. "Unique Machine States Through Output Consideration." Annual Fluid Power Res. Conf., Oklahoma State University, Paper No. P75-55 (1975).

(30) Boole, G. An Investigation of the Laws of Thought. London: Mcmillan Co., 1854; Reprinted, New York: Dover Pub., 1958.

(31) Quine, W. V. "The Problem of Simplifying Truth Functions." American Math. Monthly, Vol. 59, No. 8 (1952), 521-531.

(32) Roth, J. P. "Algebraic Topological Methods for the Synthesis of Switching Systems, I." Princeton University Press. (July 1958), 301-326 (Presented: December 29, 1955).

(33) Roth, J. P. "Minimization Over Boolean Trees." IBM Journal (November 1960), 548-558.

(34) Myers, G. E. "Model for Boolean Polynomial Simplification (POL-SIM)." (Unpub. MBA Thesis, Oklahoma State University, 1972.)

(35) Dietmeyer, D. L. Logic Design of Digital Systems, Ed. Boston, Mass.: Allyn & Bacon, Inc., 1970.

(36) Zissos, D., and F. G. Duncan. "Boolean Minimization." The Computer Journal, Vol. 16, No. 2 (March 1973), 174-179.

(37) Surjaatmadja, J. B. "A Computer Oriented Method for Boolean Simplification and Potential Hazard Elimination." Annual Fluid Power Res. Conf., Oklahoma State University, Report No. R73-2 (1973).

(38) Surjaatmadja, J. B. "TAB II-Revised Program and Users Guide for the Simplification and Static Hazard Elimination of Colossal Boolean Expressions." Annual Fluid Power Res. Conf., Oklahoma State University, Report No. R75-2 (1975).

(39) Fitch, E. C., and J. B. Surjaatmadja. Introduction to Fluid Logic, (Manuscript). Stillwater, Oklahoma: Fluid Power Press, 1975. New York: McGraw-Hill (in press).

(40) Surjaatmadja, J. B. "A Computer-Oriented Method for Complementing Boolean Expressions." Annual Fluid Power Res. Conf., Oklahoma State University, Paper No. P75-59 (1975).

(41) Surjaatmadja, J. B., and E. C. Fitch. "Logic Specifications -- Their Descriptions and Simplifications." Annual Fluid Power Res. Conf., Oklahoma State University, Paper No. P75-56 (1975).

(42) McCluskey, E. J. and H. Schorr. "Essential Multiple Output Prime Implicants." Digital Syst. Lab., Princeton University, Techn. Report No. 23 (April 1962).

(43) Bartee, T. C. "Computer Design of Multiple Output Networks." IRE Trans. on Elect. Comp., Vol. EC-10 (1961), 21-30.

(44) Schneider, P. R., and D. L. Dietmeyer. "An Algorithm for Synthesis of Multiple Output Combinational Logi." IEEE Trans. on Elect. Comp., Vol. EC-18 (1969), 117-128.

(45) Su, Y. H., and D. L. Dietmeyer. "Computer Reduction of Two Level, Multiple Output Switching Circuits," IEEE Trans. on Elect. Comp., Vol. EC-18 (1969), 58-63.

(46) Surjaatmadja, J. B. "The Transformation and Implementation of NOR Logic Circuits." Annual Fluid Power Res. Conf., Oklahoma State University, Paper No. P76-44 (1976).

APPENDIX

Eight example problems were utilized for the comparisons of the different synthesis approaches. These problems are given in an LSC format as shown in Tables XXVI-XXXIII.

TABLE XXVI

PROBLEM I

Outputs			Input States (abc)					
Z_1	Z_2	Z_3	000	100	110	101	111	010
0	0	0	(1)	2				8
0	0	1	1	(2)	3			
0	0	1		4	(3)		6	
1	0	1		(4)	3	5		
1	1	1		2		(5)	6	
1	1	1			7	5	(6)	
1	0	1			(7)		6	8
1	0	0	1					(8)

TABLE XXVII

PROBLEM II

Outputs		Input States (abc)				
Z_1	Z_2	001	000	010	101	011
1	0	(1)	2		3	
1	1		(2)	4		
0	1	6			(3)	5
0	0		7	(4)		5
1	0	6			3	(5)
1	1	(6)	8		9	
0	1	1	(7)			
1	1		(8)	4		
1	0	1			(9)	10
0	0			4	9	(10)

TABLE XXVIII

PROBLEM III

Outputs		Input States (ab)			
Z_1	Z_2	00	01	11	10
0	1	(1)	2		
0	0	3	(2)		
0	1	(3)	4		
1	1		(4)	5	
1	0			(5)	6
0	0	7			(6)
1	1	(7)	8		
0	1	9	(8)		
0	1	(9)	10		
1	1		(10)	11	
1	0			(11)	12
0	0	13			(12)
-	1	(13)	14		
0	-	3	(14)		

TABLE XXIX

PROBLEM IV

Outputs		Input States (ab)			
Z_1	Z_2	00	01	11	10
0	0	(1)	2	3	6
0	1	5	(2)		10
-	1	1		(3)	10
0	1		(4)	8	9
0	0	(5)	4	7	10
1	1		4	3	(6)
0	1	5		(7)	6
1	1	1		(8)	9
1	0		2	3	(9)
1	-		2	7	(10)

TABLE XXX

PROBLEM V

Outputs		Input States (abcd)			
Z_1	Z_2	0001	0010	0100	1000
0	1	(1)	2		3
1	1	4	(2)	5	
1	0	4		6	(3)
1	0	(4)	7		8
0	1		2	(5)	8
1	0		7	(6)	3
0	1	1	(7)	5	
1	1	4		5	(8)

TABLE XXXI

PROBLEM VI

Outputs		Input States (abcd)			
Z_1	Z_2	1000	0100	0010	0001
0	0	(1)	2		
0	0		(2)	3	
1	0		4	(3)	
1	0		(4)	5	
1	0			(5)	6
1	1			7	(6)
1	1		8	(7)	
1	1	1	(8)		

TABLE XXXII

PROBLEM VII

Outputs			Input States (abc)			
Z_1	Z_2	Z_3	000	100	010	001
1	0	0	(1)	2		
0	0	0	3	(2)		
1	0	0	(3)	4		
0	0	0	5	(4)		
0	1	0	(5)		6	
0	0	0	7		(6)	
0	1	0	(7)		8	
0	0	0	9		(8)	
0	0	1	(9)			10
0	0	0	1			(10)

TABLE XXXIII

PROBLEM VIII

Outputs		Input States (abcdef)						
Z_1	Z_2	111011	111010	011010	001010	000010	111001	111101
1	0	(1)	2					
1	0		(2)	3				
1	0			(3)	4			
1	0				(4)	5		
0	0				6	(5)		
0	0			7	(6)			
0	0		8	(7)				
0	0	9	(8)					
1	0	(9)	10					
1	0		(10)	11				
1	0			(11)	12			
0	0			13	(12)			
0	0		14	(13)				
0	0	15	(14)					
1	0	(15)	16					
1	0		(16)	17				
0	0		18	(17)				
0	0	19	(18)					
0	1	(19)					20	
0	1						(20)	21
0	0						22	(21)
0	0	1					(22)	

VITA 2

Jim Basuki Surjaatmadja

Candidate for the Degree of

Doctor of Philosophy

Thesis: A SYNTHESIS TECHNIQUE FOR THE RESOLUTION OF LARGE SCALE FLUID LOGIC SYSTEMS

Major Field: Mechanical Engineering

Biographical:

Personal Data: Born April 17th, 1945, in Malang, Indonesia; the son of Dr. Retna Wirasantosa and Rudolph Surjaatmadja.

Education: Graduate from the St. Albertus High School, Malang, Indonesia, in August, 1963; received the Sarjana I (Bachelor of Engineering) and the Sarjana (Master of Engineering) degrees from the Institute of Technology of Bandung, Bandung, Indonesia, in July, 1970 and March, 1971, respectively. Received the Master of Science in Mechanical Engineering from Oklahoma State University, in December, 1972; and completed the requirements for the degree of Doctor of Philosophy in December, 1976.

Professional Experience: Practical Training at Gruno Nasional, Ltd., Sourabaya, Indonesia and at Caltex Pacific Indonesia, Dumai, Indonesia; for six months for partial fulfillment of the Sarjana degree. Employed by IBM Indonesia as a systems engineer, 1970-1971. Graduate teaching assistant at Oklahoma State University, January-May 1972, Graduate research assistant at Oklahoma State University, from 1972 until present.

Professional Organizations: Member of the American Society of Mechanical Engineers, and the Toastmasters International.